

# 8085 Microprocessor Interfacing and Applications

EBI-8085

TEXTBOOK

595-4299-04

 **Core** Technology

 **HEATHKIT**<sup>TM</sup>  
EDUCATIONAL SYSTEMS

Prepare to succeed.<sup>TM</sup>



# **8085 Microprocessor Interfacing and Applications**

EBI-8085

TEXTBOOK

595-4299-04

Written by: Andrew C. Staugaard, Jr.  
Professor of Computer Science  
The School Of The Ozarks



**8085 MICROPROCESSOR INTERFACING AND APPLICATIONS,  
Textbook**

Copyright © 2003, 2001, 1994, 1989 by Heathkit Company, Inc., Benton Harbor, Michigan 49022. All Rights Reserved. Printed in the United States of America. Except as permitted under the United States Copyright Act of 1976, no part of this publication may be reproduced or distributed in any form or by any means, electronic or mechanical, including photocopying, recording, storage in a data base or retrieval system, or otherwise, without the prior written permission of the publisher.

ISBN 0-87119-205-5



## INTRODUCTION

The possible applications of microprocessors are almost endless and are only limited by the imagination. Just a few short years ago, it would have been unthinkable to dedicate a computer's intelligence to such common everyday items such as automobiles, appliances, toys, games, etc. Now, because of the microprocessor, just about any electromechanical device is a candidate for computer control. In fact, we have only begun to apply the microprocessor technology that exists today. The next few years will indeed be a period of microprocessor applications technology.

How can you take advantage of this applications revolution? Easy, by learning the fundamental concepts of microprocessor programming, interfacing, and applications. The **8085 Microprocessor Programming** course which is a prerequisite for this course, taught you the basics of microprocessor programming. You studied the internal register structure, or architecture, of a typical 8-bit microprocessor and learned to program it by studying its instruction set and various addressing modes. However, a microprocessor is useless by itself. It must be interfaced to memory and I/O devices to perform real world computing and applications tasks.

In this course, you will continue your study of microprocessors by learning state-of-the-art interfacing and applications concepts. You will learn how to interface memory for program storage, and I/O devices for system communication. In addition, you will learn how to apply the microprocessor to real world tasks by gaining a working knowledge analog conversion, signal conditioning, sensors, motors, control devices, and control circuits. Armed with this knowledge, you will be able to apply microprocessor technology to real world interfacing and applications tasks. This is where the action is! If you can master the fundamental interfacing and applications concepts presented in this course, you will be well on your way to taking an active part in the microprocessor applications revolution.

To perform the experiments in this course, you will need the ET-3800 Microprocessor Trainer with an 8085 CPU module. In addition, you will also need a volt-ohm-milliammeter (VOM or DVM), an oscilloscope, and some household items. All other parts such as ICs, resistors, motors, and sensors have been supplied.

Enjoy!

*Prof. Andrew C. Staugaard, Jr.*



---

## CONTENTS

INTRODUCTION.....	IV
COURSE OBJECTIVES.....	V
COURSE OUTLINE.....	VII
UNIT 1 — INTERFACING BASICS AND THE 8085 .....	1-1
UNIT 2 — MEMORY .....	2-1
UNIT 3 — PROGRAMMABLE I/O DEVICES.....	3-1
UNIT 4 — SERIAL DATA COMMUNICATIONS.....	4-1
UNIT 5 — PROGRAMMABLE TIMERS.....	5-1
UNIT 6 — ANALOG CONVERTER INTERFACING AND APPLICATIONS.....	6-1
UNIT 7 — TEMPERATURE AND OPTICAL SENSING.....	7-1
UNIT 8 — POSITION, PROXIMITY, AND FORCE SENSING.....	8-1
UNIT 9 — CONTROL DEVICES AND CIRCUITS .....	9-1
UNIT 10 — MICROPROCESSOR APPLICATIONS .....	10-1
INDEX .....	I-1



## COURSE OBJECTIVES

When you have completed this course, you will be able to:

1. Describe how to interface the 8085 with the "outside world."
2. Input and output parallel data through a Multifunction Microprocessor Support Controller (MUART).
3. Use a Multifunction Microprocessor Support Controller to provide hand-shake control of parallel I/O operations.
4. Explain the basic concepts of serial communication and how to provide both software and hardware parallel/serial conversions for a micro-computer.
5. Input and output serial data through a Multifunction Microprocessor Support Controller.
6. Describe the internal hardware structure and features of the 8256 Multifunction Microprocessor Support Controller.
7. Explain how to interface static and dynamic memory.
8. Describe the operating characteristics of a Programmable Timer Module (PTM).
9. Interface a PTM to a microcomputer system to generate timed intervals, output waveforms, and measure input waveforms.
10. Interface digital-to-analog and analog-to-digital converters to a micro-processor system.
11. Construct a microprocessor-controlled digital multimeter, or DMM.
12. Describe the component requirements of data acquisition and process control systems.
13. Compare the operating characteristics of the following temperature sensors:

- RTDs
- Thermistors
- Semiconductor junctions
- Solid state temperature sensors
- Bimetallic strips and discs
- Mercury columns

14. Construct a microprocessor-controlled thermometer.
15. Compare the operating characteristics of photoconductive, photovoltaic, and photoemissive optical sensing devices.
16. Construct a microprocessor-controlled light meter, or photometer.
17. Develop the software and hardware required to allow a microprocessor to measure the following mechanical phenomena:
  - Linear and angular position
  - Linear and angular velocity
  - Linear and angular acceleration
  - Force
18. Explain the operation of Hall-effect devices and design a Hall-effect linear position and velocity measuring circuit.
19. Explain how strain gages are used with a microprocessor to measure force.
20. Design and construct a microprocessor/stepper motor interface and control circuit.
21. Describe a typical flexible manufacturing system, or FMS.
22. Develop an understanding of how microprocessors are used in the following applications:
  - Automobiles
  - Weather monitoring systems
  - Aviation
  - Consumer products
  - Patient monitoring systems
  - Personal and business computer systems
  - Robots
  - CAD/CAM systems
  - Flexible manufacturing systems
  - Word processors
  - Copiers
  - Cash registers
  - Inventory control systems
  - Process control systems

## COURSE OUTLINE

### UNIT 1 INTERFACING BASICS AND THE 8085

Introduction

Unit Objectives

Interfacing Fundamentals

Buses

Address Decoding

3-State Buffering and Logic

Self-Test Review

Answers

8085 Interfacing

The 8085 I/O Lines

The 8085 Data Lines

The 8085 Address Lines

The 8085 Control/Status Lines

The 8085 Interrupts

The RESET interrupt.

The INTR (Interrupt Request) Interrupt

The TRAP, RST 5.5, RST 6.5, and RST 7.5 Interrupts

I/O Execution

Direct I/O

Memory-Mapped I/O

Self-Test Review

Answers

Interfacing With Switches - Data Input

Interfacing Requirements

Selecting the Switch

Detecting Contact Closure

Debouncing the Switch

Decoding the Switches

A Typical Keyboard Arrangement

The Circuit

Detecting the Key Closure

Debouncing the Key Closure

Decoding the Key Closure

Self-Test Review

Answers

Interfacing With Displays - Data Output

Driving the 7-Segment Display with the MPU

Using an Addressable Latch

Multiplexing Displays

Self-Test Review

Answers

Unit Summary

## UNIT 2 MEMORY

- Introduction
- Unit Objectives
- Interfacing to RAM
  - Static RAM Interfacing
  - Connecting RAM to the MPU
  - Dynamic RAM Interfacing
    - Dynamic RAM Refresh
    - Dynamic RAM Control
  - Connecting Dynamic RAM to the MPU
- 64K and Larger RAMs
- Self-Test Review
- Answers
- EPROM Programming and Interfacing
  - A Typical EPROM Architecture
  - Programming the EPROM
  - Self-Test Review
  - Answers
- The ETW-3800(8085) Trainer Memory Map
  - I/O Memory Map
- Unit Summary

## UNIT 3 PROGRAMMABLE I/O DEVICES

- Introduction
- Unit Objectives
- Programmable Parallel I/O Ports
  - The Programmable Parallel I/O Section
  - Internal Registers
  - Interfacing and Addressing
  - Initialization
  - Self-Test Review
  - Answers
- Using A Programmable Parallel I/O Port
  - Driving 7-Segment Displays
  - Decoding Keyboards
  - Decoding a Switch Matrix
  - Self-Test Review
  - Answers
- I/O Control Techniques
  - Interrupt Control of I/O Operations
  - Handshaking With the MUART
    - Input Handshake
    - Output Handshake
    - Configuring the MUART for Handshake Operations

- Interrupt Control Using the MUART
  - Using the 8085 *INTR* Interrupt Input
  - Using the *RST 7.5*, *RST 6.5*, *RST 5.5*, or *TRAP*
  - Interrupt Inputs
- Interrupt Enabling Within the MUART
- The *P17* and *EXTINT* Interrupts
- Polled Control of I/O Operations
- Self-Test Review
- Answers
- Unit Summary

## UNIT 4 SERIAL DATA COMMUNICATIONS

- Introduction
- Unit Objectives
- Serial Communications
  - Serial Data Formats and Standards
    - Mark/Space
    - Frequency Modulation
  - Asynchronous/Synchronous Transmission and ASCII Coding
  - Serial Communications Channeling
  - Self-Test Review
  - Answers
- Parallel/Serial Conversion
  - Software Conversion via the 8085
    - Serial Output
    - Serial Input
    - Serial I/O Timing
  - Hardware Conversion
  - Self-Test Review
  - Answers
- Serial I/O Using the 8256 MUART
  - Functional Layout and Registers
  - Registers and Addressing
    - Transmit Buffer Register Receive Buffer Register
    - Command Register 1
    - Command Register 2
    - Command Register 3
    - Status Register
  - Self-Test Review
  - Answers
- Unit Summary



## UNIT 5 PROGRAMMABLE TIMERS

- Introduction
- Unit Objectives
- Programmable Timers
  - General Programmable Timer Concepts
  - The 6840 Programmable Timer Module, or PTM
    - I/O Diagram
    - PTM Registers
      - Counters
      - Latches
      - Control Registers
      - Status Register
  - Self-Test Review
  - Answers
- How to Address and Initialize the 6840 PTM
  - 6840 PTM Interfacing and Addressing
  - PTM Initialization
  - Self-Test Review
  - Answers
- Using the PTM
  - Timer Output Mode Tasks
    - Interrupt Generation
    - Continuous Waveform Generation
    - One-Shot Pulse Generation
  - Timer Input Mode Tasks
    - Period Measurement
    - Pulse Measurement
  - Self-Test Review
  - Answers
- The Counter/Timer Section of the MUART
  - Counter versus Timer
  - Configuring the MUART Timer/Counters
  - Self-Test Review
  - Answers
- Unit Summary

## UNIT 6 ANALOG CONVERTER INTERFACING AND APPLICATIONS

- Introduction
- Unit Objectives
- Interfacing to D/A Converters
  - Self-Test Review
  - Answers

## Microprocessor Applications Using D/A Converters

- Waveform Generation

- X-Y Displays

- Programmable Gain Amplifier and Attenuator

- Motor Control and Positioning

- Process Control

  - Analog Multiplexer

  - Sample/Hold

- Self-Test Review

- Answers

## Interfacing To A/D Converters

- Interfacing and Controlling ADC Devices

  - Handshake Control of an ADC

- Interfacing and Controlling V/F Converters

  - V/F Converter Interfacing Using External Hardware Techniques

  - V/F Converter Interfacing Using Software Techniques

- Self-Test Review

- Answers

## Microprocessor Applications Using A/D converters

- Instrumentation

- Data Acquisition

  - Analog Multiplexers in a Data Acquisition System

  - Sample/Hold Circuits in a Data Acquisition System

- Analog Sensing and Industrial Control Systems

- Self-Test Review

- Answers

## Unit Summary

# UNIT 7 TEMPERATURE AND OPTICAL SENSING

- Introduction

- Unit Objectives

- Sensing Temperature

  - Thermoresistive Temperature Sensing Devices

    - Resistance Temperature Detectors, or RTDs

      - Construction and Operation

      - Characteristics

      - Application Circuits

    - Thermistors

      - Construction and Operation

      - Characteristics

      - Application Circuits

  - Thermoelectric Temperature Sensing Devices

    - Construction and Operation

    - Characteristics

    - Application Circuits

- Semiconductor Junction Devices
- Thermoswitches
  - Construction, Operation, and Characteristics
  - Liquid Expansion Thermoswitches
- Self-Test Review
- Answers
- Optical Sensing
  - Photoconductive Devices
    - Construction and Operation
    - Characteristics
    - Application Circuits
  - Photovoltaic Devices and Photodiodes
    - The Photovoltaic Mode
    - The Photoconductive Mode
    - Characteristics
    - Application Circuits
  - Phototransistors
    - Construction and Operation
    - Characteristics
    - Application Circuits
  - Integrated Optical Sensing Devices
    - Construction and Operation
    - Characteristics
    - Application Circuits
- Self-Test Review
- Answers
- Unit Summary

## **UNIT 8 POSITION, PROXIMITY, AND FORCE SENSING**

- Introduction
- Unit Objectives
- Sensing Position and Proximity
  - Potentiometric Position Sensors
  - Capacitive Position and Proximity Sensors
    - Sensing Position
    - Sensing Proximity
  - Inductive Position and Proximity Sensors
    - Sensing Position
      - The LVDT
      - The RVDT
    - Sensing Proximity
  - Magnetic Position and Proximity Sensors
    - The Magnetic Reed Switch
    - Hall-effect Devices
    - Sensing Position

**Force Sensing****Force****Resistive Strain Gages****Construction and Operation****Characteristics****Application Circuits****Semiconductor Strain Gages****Construction and Operation****Characteristics****Application Circuits****Piezoelectric Strain Gages****Load Cells****Construction and Operation****Characteristics****Application Circuits****Self-Test Review****Answers****Unit Summary****UNIT 9 CONTROL DEVICES AND CIRCUITS****Introduction****Unit Objectives****Electronic Control Devices and Circuits****Open Collector Drivers****Bipolar Power Transistors****Bipolar Transistor Arrays****Power MOSFETs****Peripheral Power Drivers****Optocouplers****Optoisolator Characteristics****Thyristors****SCRs****TRIACs****SCR and TRIAC Control Circuits****Relay and Solenoid Actuators****Solid-State Relays****Electro-Mechanical Relays****Solenoids****Self-Test Review****Answers****MPU Control of DC Motors****Controlling Permanent Magnet DC Motors****Using a D/A Converter for Motor Control****Using Pulse-Width Modulation for Motor Control**

- Controlling Wound-Field DC Motors
- Stepper Motors
  - Bipolar Permanent Magnet Stepper
  - Bifilar, or Unipolar, Motor
- Controlling Stepper Motors
  - Bipolar Control
  - Unipolar Control
  - The MPU Interface
- Self-Test Review
- Answers
- Unit Summary

## UNIT 10 MICROPROCESSOR APPLICATIONS

- Introduction
- Unit Objectives
- Consumer Applications
  - The Automobile
  - Appliances
  - A Weather Computer
  - Other Consumer Products That Think
  - Personal Computers
  - Self-Test Review
  - Answers
- Industrial Applications
  - Production Industry Applications
    - Robotics
      - The Robot System
  - CAD/CAM
  - Flexible Manufacturing Systems
  - Commercial Aviation Applications
  - Medical Applications
  - Self-Test Review
  - Answers
- Business Applications
  - Small Business Systems and Word Processors
  - Copiers
  - Cash Registers and Inventory Control
  - Self-Test Review
  - Answers
- Unit Summary



*Unit 1*

**INTERFACING BASICS AND THE 8085**

## CONTENTS

Introduction.....	1-3
Unit Objectives.....	1-4
Interfacing Fundamentals.....	1-5
Buses.....	1-5
Address Decoding.....	1-7
3-State Buffering and Logic.....	1-11
Self-Test Review.....	1-15
Answers.....	1-16
8085 Interfacing.....	1-18
The 8085 I/O Lines.....	1-18
The 8085 Data Lines.....	1-19
The 8085 Address Lines.....	1-20
The 8085 Control/Status Lines.....	1-22
The 8085 Interrupts.....	1-23
The RESET Interrupt.....	1-24
The <i>INTR</i> (Interrupt Request) Interrupt.....	1-24
The <i>TRAP</i> , <i>RST 5.5</i> , <i>RST 6.5</i> , and <i>RST 7.5</i> Interrupts.....	1-28
I/O Execution.....	1-31
Direct I/O.....	1-32
Memory-Mapped I/O.....	1-33
Self-Test Review.....	1-36
Answers.....	1-37
Interfacing with Switches — Data Input.....	1-38
Interfacing Requirements.....	1-38
Selecting the Switch.....	1-38
Detecting Contact Closure.....	1-39
Debouncing the Switch.....	1-40
Decoding the Switches.....	1-40
A Typical Keyboard Arrangement.....	1-41
The Circuit.....	1-42
Detecting Key Closure.....	1-43
Debouncing Key Closure.....	1-45
Decoding Key Closure.....	1-46
Self-Test Review.....	1-50
Answers.....	1-51
Interfacing with Displays — Data Output.....	1-52
Driving the 7-Segment Display With the MPU.....	1-52
Using an Addressable Latch.....	1-57
Multiplexing Displays.....	1-61
Self-Test Review.....	1-64
Answers.....	1-65
Unit Summary.....	1-66

## ***Unit 1***

# **INTERFACING BASICS AND THE 8085**

## **INTRODUCTION**

In this unit, you will begin your learning journey through this comprehensive course. The emphasis in the first section of this unit will be on teaching you those concepts that are fundamental to all interfacing tasks. These concepts include address decoding, 3-state buffering, latching and timing. Make sure that you understand the ideas presented here, because they will provide a foundation for the rest of the course.

In the second section of this unit you will learn about the features of the 8085 and how it can be interfaced to the outside world. Be sure that you understand both the direct I/O and memory-mapped I/O method of 8085 interfacing.

Finally, the last two sections of this unit will show you how to interface simple input and output devices, such as switches and displays, to the MPU. Again, the simple ideas presented here will be fundamental to the more advanced concepts presented later in the course. Now, let's begin our journey.

## UNIT OBJECTIVES

1. Explain the bus structure of a microprocessor system.
2. Design both SSI and MSI address decoder circuits.
3. Define 3-state logic and latching — explain the need for these functions in microprocessor interfacing.
4. Describe the 8085 I/O lines.
5. Design a circuit to demultiplex the 8085 low address bus from the data bus.
6. Explain how the 8085 can employ both direct I/O and memory-mapped I/O to communicate with I/O devices.
7. Design both the hardware and software required for switch and keyboard interfacing.
8. Explain how the MPU can be programmed to eliminate switch contact bounce.
9. Explain the operation of a program that detects contact closure of switches, provides for switch debouncing, and decodes a simple keyboard.
10. Describe different methods of 7-segment display interfacing.

## INTERFACING FUNDAMENTALS

Before getting into specific interfacing examples, we must discuss some fundamental concepts that are common to all interfacing tasks. This includes the concept of a bus and the need for address decoding and 3-state logic.

### Buses

In computer jargon, a bus is generally defined as a group of conductors over which information is transferred from one place to another. In many cases, the information can originate from any one of several sources and can be transferred to any one of several destinations. When data is transferred in only one direction the bus/data is referred to as **unidirectional**. Moreover, on some buses, information can be transferred in either of two directions. These are called **bi-directional** buses.

Figure 1-1 shows the data bus arrangement in a typical microcomputer application. Generally in this type of system, all data transfers involve the MPU. Thus, data can be transferred in either direction between RAM and the MPU. However, other data transfers are one way only. Data can be transferred from ROM or the input buffer to the MPU. Also, data can be transferred from the MPU to the output latches.

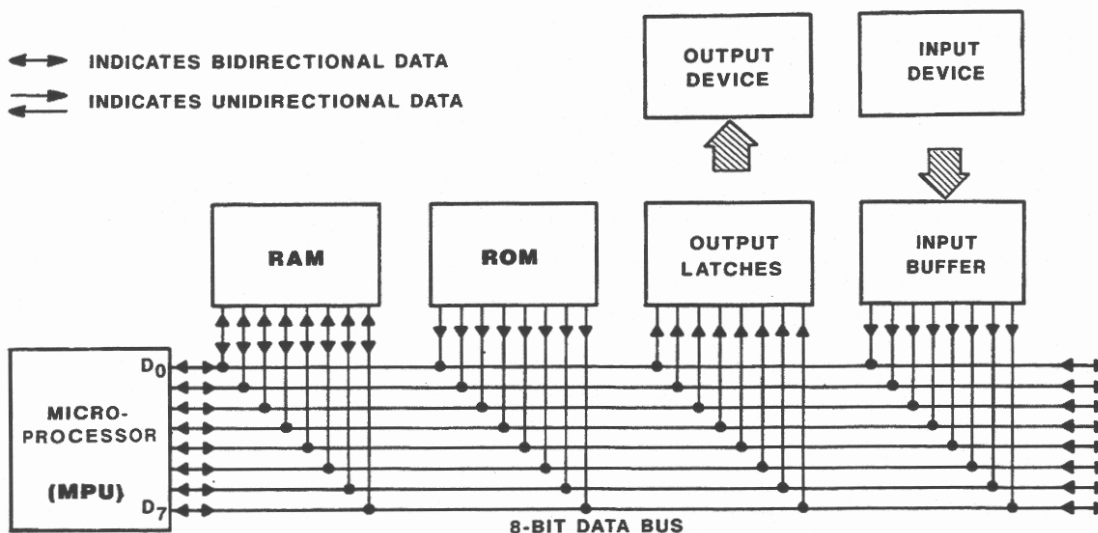


Figure 1-1  
Typical Data Bus Arrangement.



In an arrangement like this, two problems arise. First, we must insure that only one data transfer is attempted at any given time. This is done by assigning each destination or source a different address. For example, the RAM, ROM, output latches, and input buffers all have one or more chip enable pins. The proper logic levels on these pins will select or activate the respective circuit when a given address appears on the address bus. By assigning each circuit a different address, we insure that only one circuit at a time is enabled.

Figure 1-2 shows the addressing capability added to the block diagram. An address decoder is added for each circuit. The inputs to the address decoders come from the MPU via the address bus. The outputs go to the chip enable lines of the various circuits. Since only one address can appear on the address bus at any given instant, only one of the external circuits will be enabled at a time.

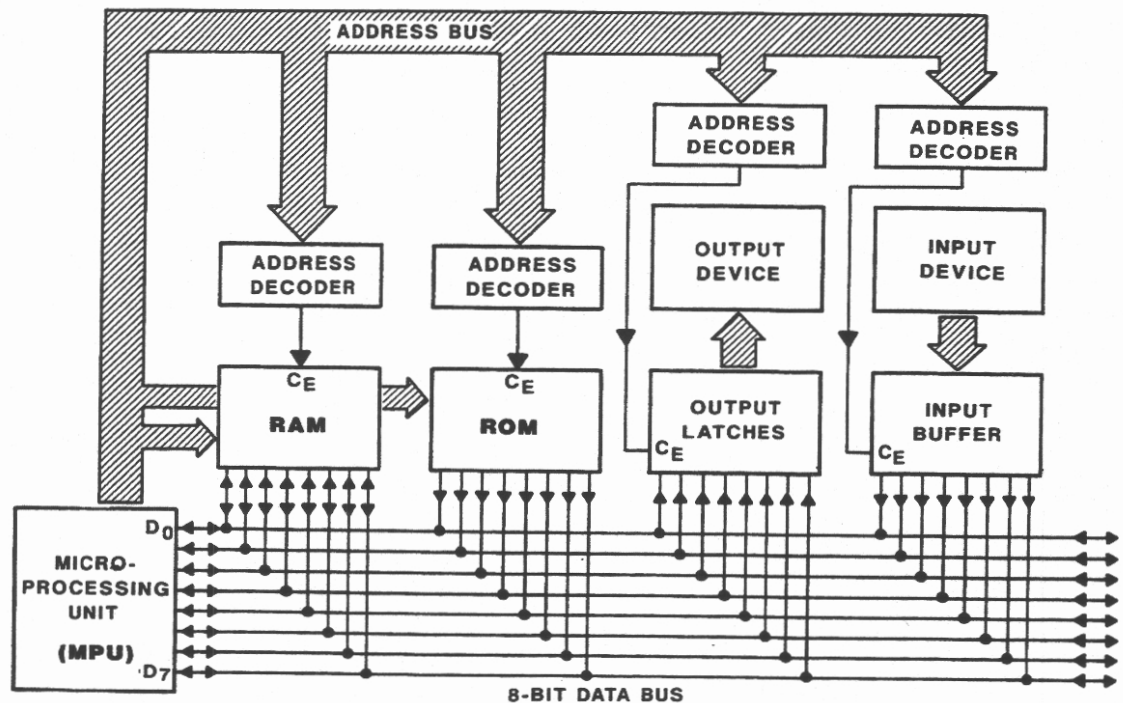


Figure 1-2  
Adding the Address Capability.

The memories are assigned many addresses because each byte must have its own address. For example, if a  $512_{10}$  byte RAM is used, it might be assigned addresses  $0000_{16}$  through  $01FF_{16}$ . When any one of these addresses appear on the address bus, the RAM is selected via its chip enable line. Notice that a portion of the address bus connects to the address decoder and another portion directly to the RAM. The address bus lines that are connected to the decoder are used to enable the RAM chip via the decoder, while those that are connected directly to the RAM are used to select an individual byte within the RAM.

The ROM is assigned a range of addresses. If a  $1024_{10}$  byte ROM is used, it may be assigned addresses  $FC00_{16}$  through  $FFFF_{16}$ . The ROM must be enabled whenever any of these addresses appear on the address bus.

The output latch and input buffers are also assigned their own addresses. Thus, the MPU can communicate with any one of the external circuits simply by placing the proper address on the address bus. Again, notice the use address decoders to "recognize" a given address when it appears on the address bus and subsequently enable the output latch or input buffer. Let's take a closer look inside the address decoder.

## Address Decoding

When data is to be read from external memory or peripheral I/O devices, the MPU must know where to get the data. Likewise, when data is to be written, the MPU must know where to send the data. As you know, this is the job of the address bus. It indicates *where* data is to be read from or written to within the system. Just as your house has an address, each memory location and peripheral I/O device in the system has an address assignment. Addresses are always generated by the MPU on the one-way address bus.

The major application of address decoders in any computer system is for decoding addresses on the address bus. A typical 8-bit microprocessor can generate 65,536, or 64K, addresses. Now, let's fantasize a "bit." Suppose that *you* are an I/O device and one of these addresses has been assigned to you. The MPU performs a data transfer operation and places an address on the address bus to indicate what memory location the data is to be read from or written to. This creates a major dilemma. You must know if the address on the address bus is yours so that you can send or receive the data via the data bus. This is the job of an **address decoder**: to signal you when your address appears on the bus.

Granted, the above analogy is an over-simplification, but it clearly illustrates the function of an address decoder in a computer system. The decoder must detect a given address or set of addresses on the address bus, and signal the proper memory and/or peripheral I/O devices for the data transfer operation.

You saw the use of an address decoder in Figure 1-2. An address decoder can be a series of logic gates configured to recognize a particular address or it can be a decoding chip that makes the logic design less complicated.

Now, let's design an address decoder that will recognize a given address. Suppose our I/O device is assigned address  $5000_{16}$ . Our decoder circuit must be capable of detecting this address and then generate an enabling signal. A typical logic circuit that could perform this function is shown in Figure 1-3. The *decoding chart* at the bottom of Figure 1-3 indicates the necessary address bit status for address  $5000_{16}$ . Note that we are decoding *all* of the address lines such that the *only* address that will generate a logic 1 chip enable signal is address  $5000_{16}$ . All other addresses will cause a logic 0 to appear at the output of the decoder circuit. Thus, the address decoder "recognizes" address  $5000_{16}$  and no other addresses. When all of the address lines are decoded in this manner, we say the address is **fully decoded**. The "trick" used to design the decoder circuit is to NOR those address lines that will be a logic 0 for the decoded address and AND those lines that will be a logic 1. Of course, the outputs of the NOR gates must also be ANDed along with the logic 1 address lines.

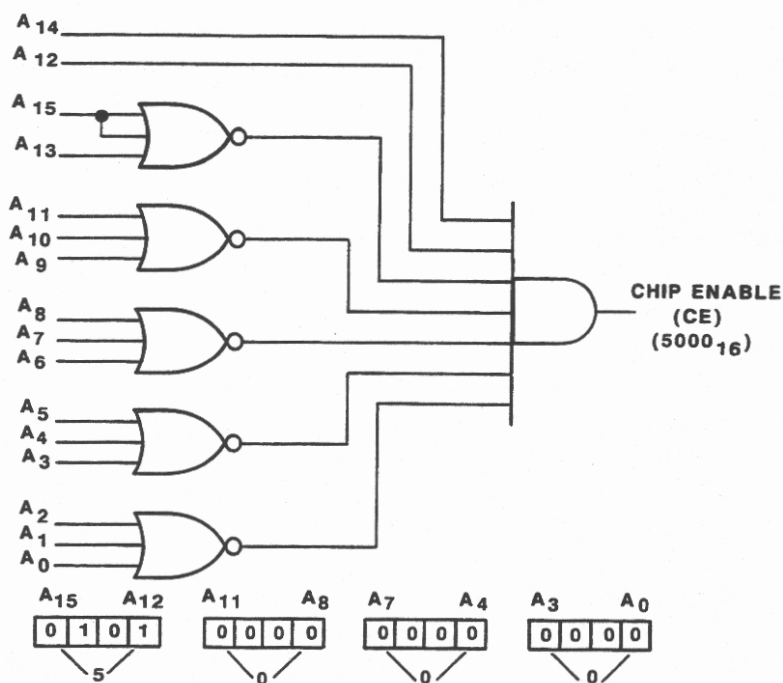


Figure 1-3

A Full Decoding Circuit for Address  $5000_{16}$

Decodes All the Address Bus Lines.

Here's how it works. The NOR gates will only produce logic 1 outputs when all their input lines are a logic 0. Thus, all the address lines that are at a logic 0 state for address  $5000_{16}$  are applied to the NOR gate inputs. Next, the AND gate will only generate a logic 1 output when all of its inputs are at a logic 1 state. Consequently, all the address lines that are at a logic 1 state for address  $5000_{16}$  ( $A_{14}$  and  $A_{12}$ ) are applied, along with the logic 1 NOR gate outputs, to the AND gate input lines. This assures that address  $5000_{16}$  is the only address that will produce a logic 1 chip enable (CE) signal on the decoder output line.

How would the decoder design change if it must generate a logic 0 chip enable signal ( $\overline{CE}$ ) rather than a logic 1 signal ( $CE$ )? You're right, you must place an inverter on the circuit output line. Thus, simply replace the AND gate with a NAND gate.

Another way to decode the same address is shown in Figure 1-4. Here we are only decoding *part* of the address bus. This type of decoding is referred to as **partial decoding**. In this example we are only decoding the upper eight bits of the address bus. The advantage of this decoding method is that fewer logic gates are required. However, the disadvantage is that any address from  $5000_{16}$  through  $50FF_{16}$  will cause an enable condition. This is the price we must pay to reduce the hardware required. Because we usually have many addresses available to use, it's no problem to allocate addresses 5000 through 50FF to one I/O device. In Figure 1-4, we show the lower half of the decoding chart filled with X's, indicating *don't care* states, either logic 1 or logic 0. In other words, we "don't care" what logic is present on these address lines, since they are not being decoded.

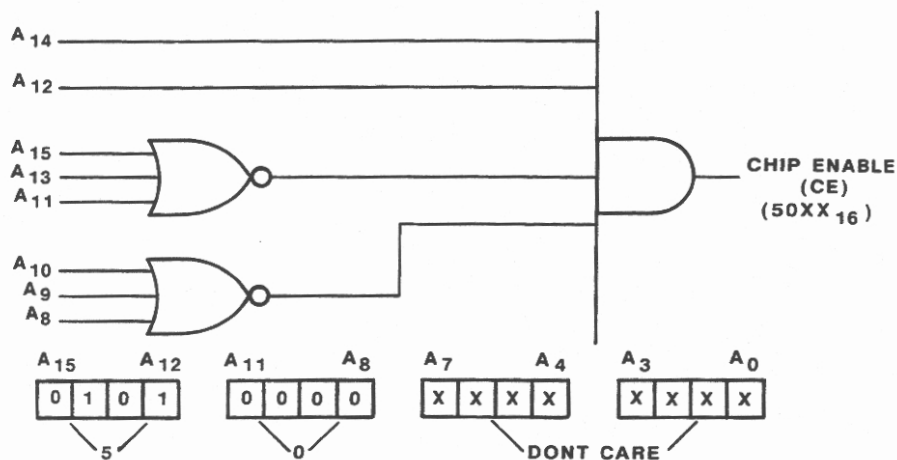


Figure 1-4

A partial decoding circuit for address  $5000_{16}$   
only decodes part of the address lines.

Another way to decode addresses is by using MSI (Medium Scale Integration) decoder chips. Many microprocessors are capable of generating 64K addresses. As a result, to detect a given address, a decoder must be designed to recognize one of 65,536 possible addresses. In other words, the design task requires a 1-of-64K decoder. If such a device is not commercially available, you must combine smaller MSI decoder chips to accomplish the task.

A 1-of-64K decoder can be designed using four 1-of-16 decoders as shown in Figure 1-5. Here, observe that the 16-bit address bus lines are labeled  $A_0$  through  $A_{15}$  and have been divided-up into four groups of four bits each. Each group of four bits is assigned to the input of a single decoder as shown.

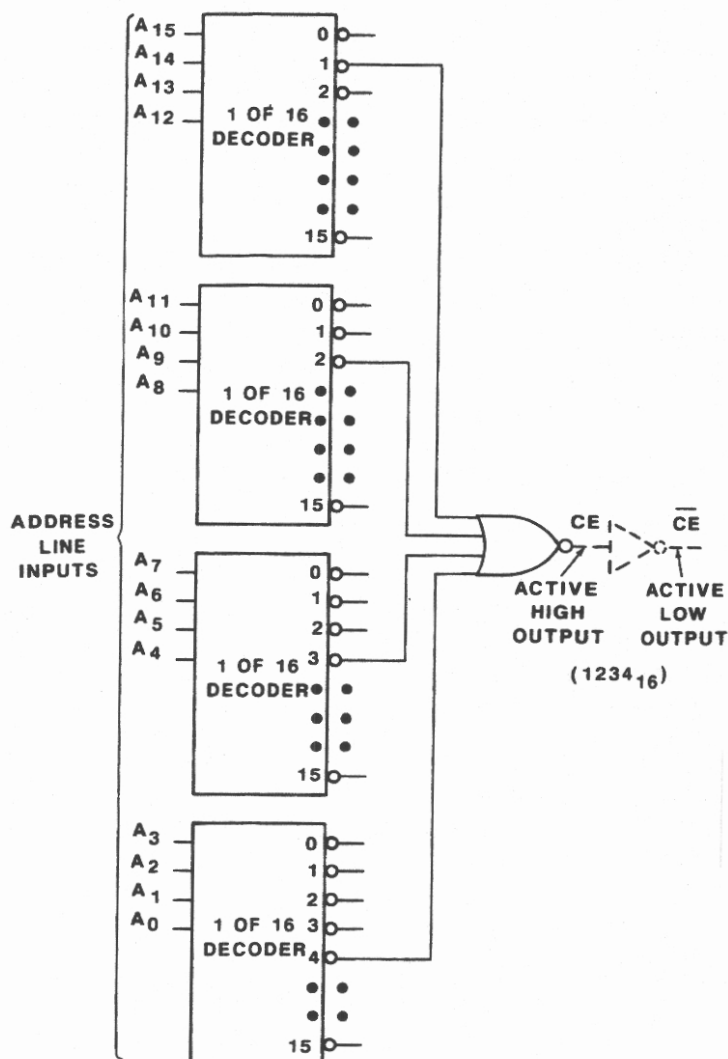


Figure 1-5

A 16-Bit address bus can be completely decoded using four 1-of-16 MSI decoder ICs.

To decode a given address you first need to convert the address to hexadecimal. This becomes apparent if you recall that a 1-of-16 decoder is a binary-to-hex decoder. Once you have the hexadecimal equivalent of the address, simply connect the corresponding decoder output lines to the input of a 4-bit NOR gate to generate a logic 1 signal or a 4-bit OR gate to generate a logic 0 chip enable signal.

For example, suppose you wish to decode hexadecimal address  $1234_{16}$ . The MPU generates the binary equivalent of this, or 0001 0010 0011 0100, on its sixteen address bus lines  $A_0$  through  $A_{15}$ . Consequently, when this address is present, the most significant decoder at the top of Figure 1-5 generates a logic 0 on output line 1, the next decoder generates a logic 0 on output line 2, the third decoder activates line 3, and the least significant decoder at the bottom of Figure 1-5 activates line 4. If these decoder output lines are used to drive a 4-bit NOR gate, the gate output will only be high when hexadecimal address  $1234_{16}$  is present on the address bus. For all other addresses, the NOR gate output will be a logic 0. Of course, the output logic could be reversed by adding an inverter to the NOR gate output, thus creating a 4-bit OR gate as shown by the dotted lines in the figure.

It is important to note that the four decoders in Figure 1-5 can be used to decode all of the 64K addresses by simply NORing different decoder outputs. Consequently, this is full decoding, since all of the address bus lines are being decoded. Of course, you can save decoder logic by only decoding part of the address bus. For instance, you can save two decoder ICs by only decoding the upper eight address lines ( $A_8 - A_{15}$ ). However, the price you must pay for this partial decoding is that an entire range of addresses will cause an enable signal to be generated. In our example, if you only decode the upper eight address lines using two decoder ICs and a 2-bit NOR gate, a logic 1 enable signal will be generated for 256 unique addresses, from  $1200_{16}$  through  $12FF_{16}$ .

### 3-State Buffering and Logic

The second problem of interfacing to a microprocessor bus system is more fundamental. It arises because of the basic 2-state nature of digital logic circuits. Recall that the output of a standard logic gate will always be either logic 1 (high) or logic 0 (low). The problem is: Which state should the outputs of the circuits that are connected to the data bus assume when they are not selected? Regardless of which state they assume, they interfere with the output of the enabled circuit. For example, if the output of the disabled circuits assume a high state, they interfere with the low output of the enabled circuit. In other words, one circuit tries to pull the bus line high while the other is trying to force it low.

In the past, this problem has been overcome by using gates with open collector outputs. While open collector devices could be used to solve this problem in microprocessors, an entirely different approach is most often used. To understand how this problem is overcome, we must discuss 3-state logic.



As the name implies, 3-state logic devices have a unique third state in addition to the normal 1 and 0 output. Figure 1-6 compares a standard non-inverting buffer with a 3-state, non-inverting buffer.

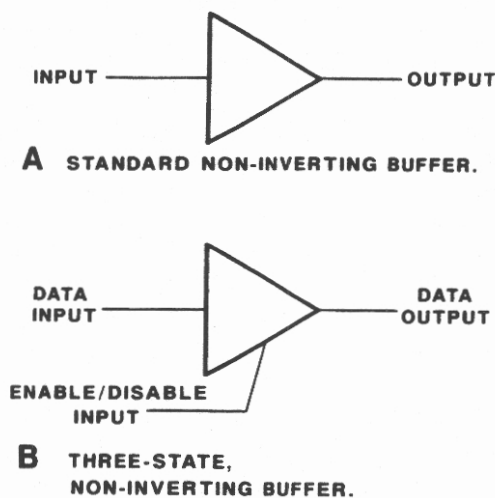


Figure 1-6

Comparison of standard and 3-state buffers.

Recall that a non-inverting buffer increases the current drive of its input signal without changing the logic levels in any way. Thus, the output may be able to drive ten times as many gates as the input. The standard buffer has one input and one output. The output always assumes the same logic level as the input. Because the input must be either 1 or 0, the output must be the same.

By contrast, the 3-state buffer has two inputs. In addition to the normal data input, the buffer has an enable/disable input. This input may be either logic 1 or logic 0 depending upon whether we wish to enable or disable the buffer. The buffer shown in Figure 1-6B is enabled by applying a logic 1 to the enable/disable input.

When enabled, the 3-state buffer acts exactly like the standard buffer. The output will assume the same logic level as the data input.

The 3-state buffer is disabled by applying a logic 0 to the enable/disable input. When disabled, the output assumes a very high impedance state that is neither logic 1 nor logic 0. While in this *high impedance* state, the output can be assumed to be disconnected from the rest of the circuit. That is, when the buffer is disabled, its output will not interfere with the circuits to which it is connected.

There are many different types of 3-state devices available. Figure 1-7 shows four different types of 3-state buffers. The buffer shown in Figure 1-7A is the same as that described above. It does not invert and is enabled by a logic 1. Notice that the buffer shown in Figure 1-7B has a small circle at the enable/disable input. This means that the buffer is enabled by a logic 0 and disabled by a logic 1.

Figure 1-7C and D show inverting buffers. The first is enabled by a logic 1; the second is enabled by a logic 0.

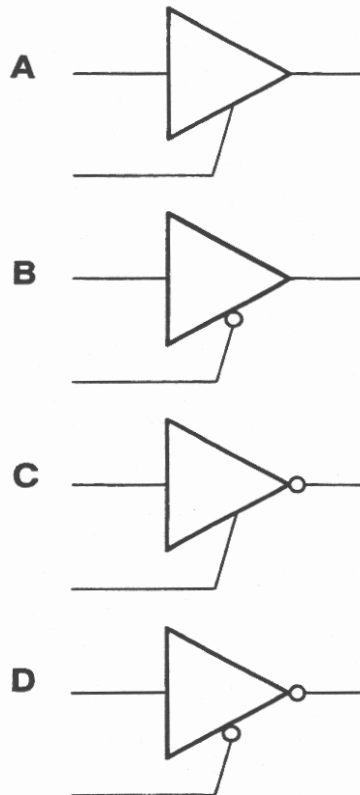


Figure 1-7  
Four types of 3-state buffers.

Generally, four or more 3-state buffers are included in a single integrated circuit. Figure 1-8 shows the 74125 type TTL IC. It contains four 3-state buffers in a single 14-pin dual-in-line package.

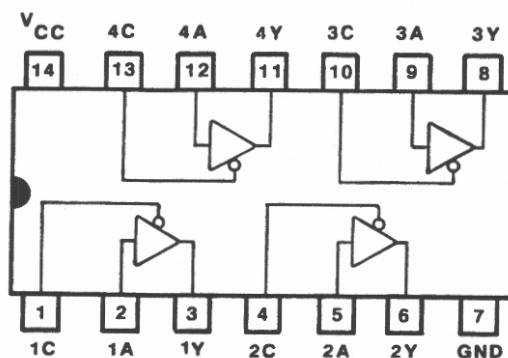


Figure 1-8

The 74125 IC contains four 3-state buffers in a single 14-pin package.

Figure 1-9 shows eight 3-state buffers in a single 20-pin package. The four lower buffers are enabled by a logic 0 at pin 1 of the IC. The four upper buffers are enabled by a logic 1 at pin 19. The input buffer shown earlier in Figures 1-1 and 1-2 could use this type of IC. Buffers of this type are often called **bus extenders**, **line drivers**, **line receivers**, etc., depending on how they are used.

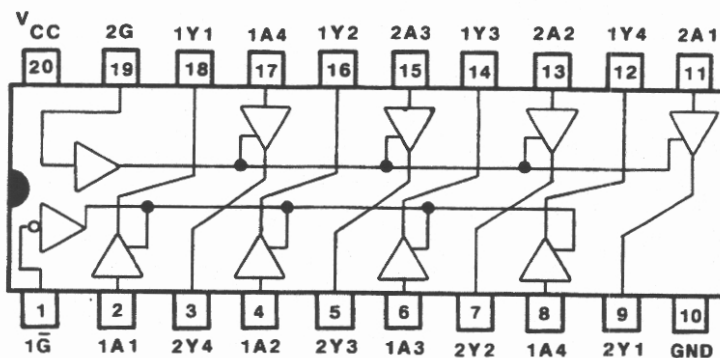


Figure 1-9

The 74LS241 contains eight 3-state non-inverting buffers.

While many different forms of 3-state buffers are available, many microprocessor support circuits do not require separate 3-state buffers. Most RAMs and ROMs have their own 3-state buffers built in. Thus, any time the RAM or ROM is not selected, it automatically goes to its third state. In this state, the outputs are said to be off, disconnected, disabled, floating, or in their high impedance state.

## Self-Test Review

1. What is the function of an address decoder in a microprocessor bus system?
2. Explain the difference between full and partial decoding. What is the advantage and disadvantage of partial decoding?
3. Design an address decoder using two 1-of-16 decoders to partially decode address  $84FE_{16}$ .
4. What range of addresses will the address decoder in question 3 respond to?
5. What is the purpose of 3-state logic in a microprocessor bus system?
6. What is a 3-state logic gate?
7. How is the non-inverting buffer shown in Figure 1-7A switched to its high impedance state?

## Answers

1. The function of an address decoder is to monitor the address bus and generate a chip enable signal when a given address or group of addresses appear on the bus.
2. Full decoding means that all the address lines are being decoded by the address decoder. With partial decoding, only part of the address bus lines are being decoded. Partial decoding saves hardware logic, but increases the number of addresses to which the decoder will respond.
3. See Figure 1-10. Here, the upper eight address lines ( $A_8 - A_{15}$ ) are being decoded by two 1-of-16 decoders. Output line 8 of decoder #1 is NORed with output line 4 of decoder #2. Thus, the NOR gate will generate a logic 1 enable signal when a hex address of  $84XX_{16}$  appears on the address bus. (Note: The X's are don't care states, since the lower half of the address bus is not being decoded).

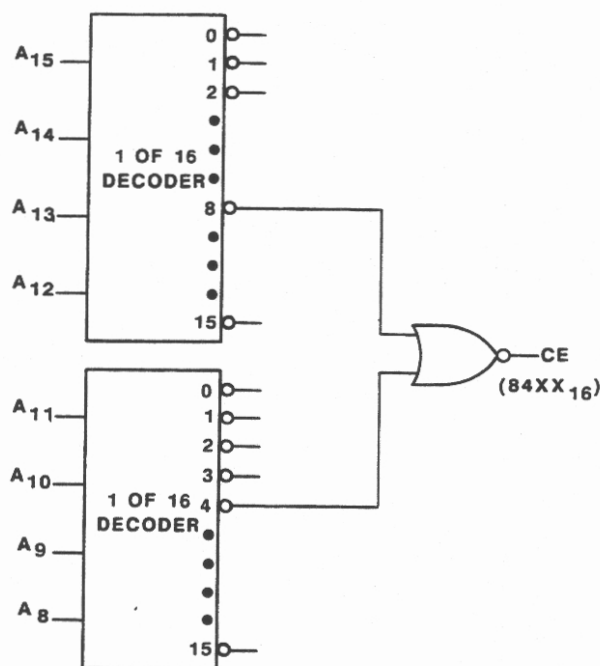


Figure 1-10

Decoder circuit for question 3.

4. Any address from  $8400_{16}$  through  $84FF_{16}$ .
5. The purpose of 3-state logic in a microprocessor bus system is to isolate the various registers within the system from the bus structure. This allows the various registers to be connected or disconnected from the bus structure as required by the MPU to perform an operation.

6. A gate which has an off state in addition to the normal logic 1 and logic 0 output states.
7. By applying a logic 0 to the enable/disable input.

## 8085 INTERFACING

The ETW-3800 Trainer on which you will be performing the experiments employs an 8085 microprocessor in its accompanying CPU module. The 8085 is manufactured by Intel and was designed as an *upward compatible* successor to Intel's popular 8080 microprocessor. The 8085 is a versatile 8-bit processor that provides an excellent vehicle for learning the basics of microprocessor interfacing and applications. Although there are now 16-bit and 32-bit microprocessors on the market, the interfacing basics that you will learn for the 8085 can be easily extended to accommodate these higher performance micros.

In this section, you will first learn about the 8085 I/O lines. Then, you will see how these lines are used to interface the 8085 with the outside world.

### The 8085 I/O Lines

The I/O lines of any MPU can be functionally divided into three categories: *data*, *address*, and *control/status*. In fact, these three I/O line categories form the data, address, and control/status buses of any microprocessor-based system as is illustrated in Figure 1-11. The data bus is used to transfer information between the MPU and memory or peripheral I/O devices. Notice from Figure 1-11 that the data bus is a bidirectional bus. The logic signals on this bus tell the system *what* information is being transferred.

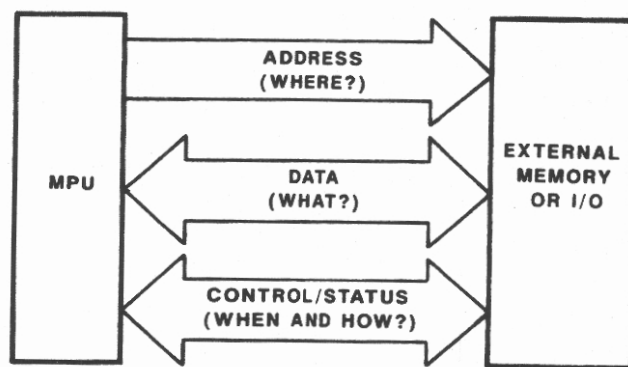


Figure 1-11

The I/O lines of any MPU can be functionally divided into three categories: address, data, and control/status.

The address bus is used by the MPU to tell the system *where* the data information is to be transferred to, or transferred from. Notice that the address bus is a unidirectional bus from the MPU to memory and peripheral devices.

Finally, the signals on the control/status bus are used to synchronize the data transfer by telling the system *when* and *how* the data is to be transferred.

Our task now is learn about those 8085 I/O lines that make-up this bus structure. A pinout diagram of the 8085 is provided in Figure 1-12. Refer to this diagram during the subsequent discussion. We will begin with the data bus.

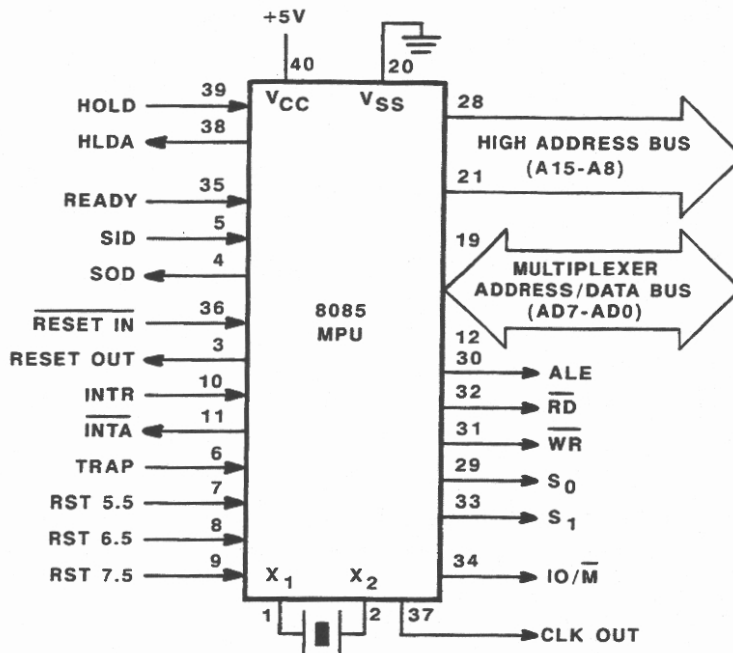


Figure 1-12  
The 8085 I/O lines.

## THE 8085 DATA LINES

Since the 8085 is an 8-bit microprocessor, it has eight data I/O lines that make-up its data bus. Notice that the data lines are bidirectional, meaning that data can be input to the MPU or output from the MPU via these data lines. An input operation is called a *read* (the microprocessor is *reading* data), while an output operation is referred to as a *write* (the microprocessor is *writing* data) operation. The eight data bus lines of the 8085 are labeled  $AD_0$  through  $AD_7$ . Data line  $AD_0$  provides the least significant data bit, while  $AD_7$  provides the most significant bit.



## THE 8085 ADDRESS LINES

There are 16 address lines that make-up the 8085 address bus. With its 16 address lines, the 8085 is capable of generating  $2^{16}$ , or 65,536, unique addresses. In computer jargon this number is referred to as 64K, where K stands for the value 1024. Thus, we say that the 8085 has a 64K address space.

You will note from Figure 1-12 that the sixteen address bus lines are divided into two halves: the high order address bus consisting of address lines  $A_8$  through  $A_{15}$  and the low order address bus consisting of line  $AD_0$  through  $AD_7$ . You are probably asking yourself: "How can this be?" Earlier you learned that lines  $AD_0$  through  $AD_7$  were the 8085 data lines. Now we are saying that they form the low order address bus. The fact is that these lines serve a **dual purpose** as both the data bus and low order address bus.

When the 8085 executes an instruction, lines  $AD_0$  through  $AD_7$  serve as the low order address bus lines during the early part of the instruction cycle. Then, during the later part of the instruction cycle, lines  $AD_0$  through  $AD_7$  serve as data lines. Thus, there is no ambiguity. At one point in time these lines are address lines, while at another point in time they act as data lines. Thus, we say that lines  $AD_0$  through  $AD_7$  provide a *multiplexed address/data bus*. This technique is commonly used in not only the 8085, but other processors as well, for the following reason.

Multiplexing the low order address bus with the data bus saves I/O lines (pins) on the MPU. However, there is one small problem: In order for the low order address bus to be synchronized with the high order address bus, the multiplexed address/data bus must be demultiplexed. In other words, the high order address signals must be separated from the data signals. This requires an external 8-bit latch as shown in Figure 1-13A. Here, the latch inputs are the address/data lines. The latch outputs are the lower order address bus lines. The associated timing diagram in Figure 1-13B shows how it works.

During the first clock (CLK) cycle (T1), both the high and low order address information is generated by the MPU. At the same time, an output control line called *ALE* (Address Latch Enable) goes high. Notice in Figure 1-13A that the *ALE* control line is connected to the *Enable* input of the latch. When *ALE* goes high, the lower order address information appears on the latch output lines. When *ALE* goes back low this address information is **latched** and remains on the latch output lines.

During the next two clock cycles (T2 and T3), **data** is transferred by the MPU via the  $AD_0$  -  $AD_7$  lines. Now there is no ambiguity since the low order address information has already been demultiplexed via the latch and lines  $AD_0$  -  $AD_7$  can now serve as data bus lines.

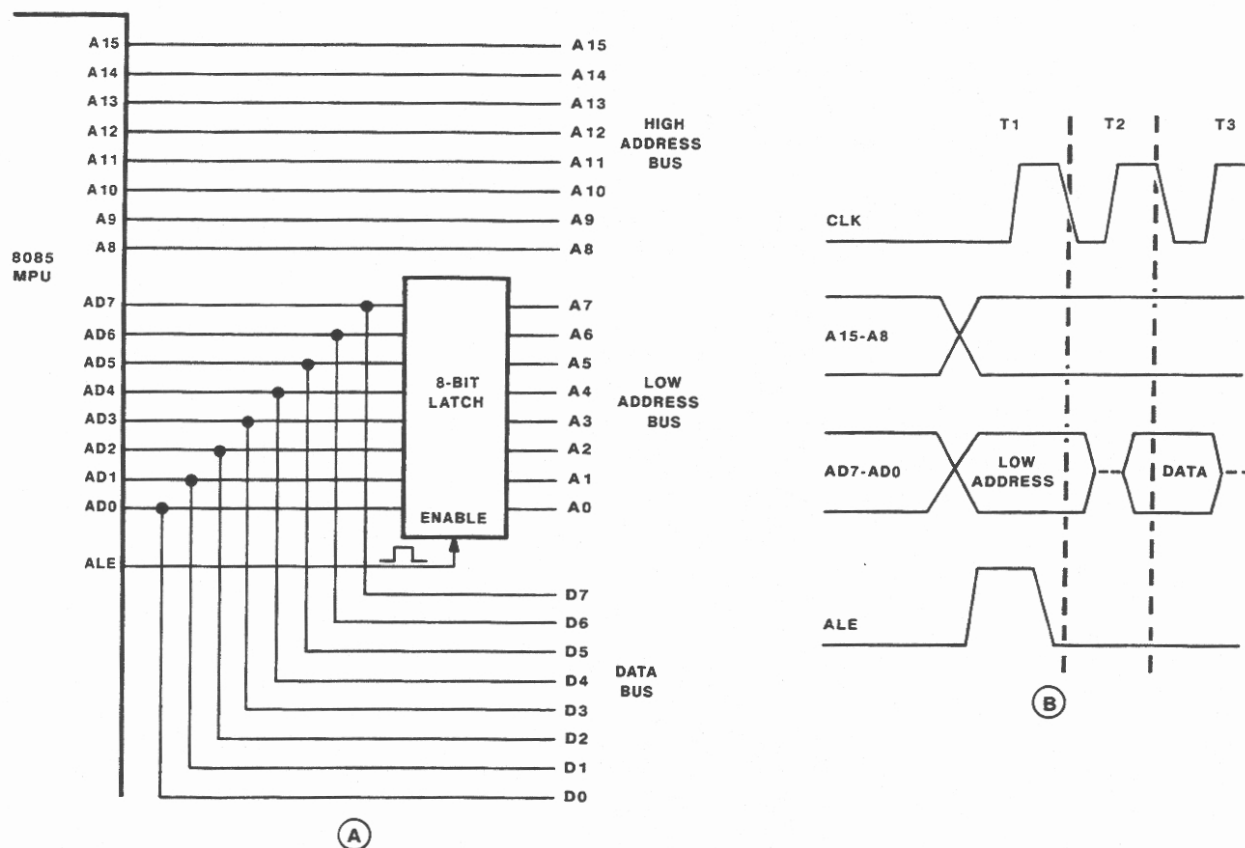


Figure 1-13

Demultiplexing the address/data bus: (a) circuit and (b) timing diagram.

## THE 8085 CONTROL/STATUS LINES

The remaining I/O lines in Figure 1-12 form the 8085 control and status bus. What follows is a brief description of each line. You will learn more about them when you see how they are used for interfacing in subsequent units.

**ALE (Address Latch Enable):** This is an active high output signal that is used to indicate when the multiplexed address/data bus ( $AD_0 - AD_7$ ) contains address information. You just learned how the *ALE* control line is used to demultiplex the address/data bus.

**$\overline{RD}$  (Read):** This is an active low output control signal that indicates to memory and peripheral devices that the 8085 is performing a read operation.

**$\overline{WR}$  (Write):** This is an active low output control signal that indicates to memory and peripheral devices that the 8085 is performing a write operation.

**$S_0$ ,  $S_1$ , and  $IO/\overline{M}$ :** These are output status lines that are used to identify various operations to external peripheral devices. Table 1-1 lists the various MPU status indications provided by these lines. Of particular importance for interfacing is the  $IO/\overline{M}$  line. You will see this shortly.

Table 1-1  
8085 Machine Cycle Status

OPERATION	$IO/\overline{M}$	$S_1$	$S_0$
Memory Write	0	0	1
Memory Read	0	1	0
I/O Write	1	0	1
I/O Read	1	1	0
Opcode Fetch	0	1	1
Interrupt Ack	1	1	1
Halt	Z	0	0
Hold	Z	X	X
Reset	Z	X	X

Key: Z = High Impedance State  
X = Don't Care State

**HOLD:** This is an active high input line that can be used by external devices to gain control of the system address and data buses. When the 8085 receives a *HOLD* signal, it will relinquish the address and data buses as soon as the current bus transfer is complete. The 8085 can regain control of the bus structure after the *HOLD* signal is removed. This type of *HOLD* operation is generally used with *DMA (Direct Memory Access)* to permit a peripheral I/O device to access memory directly, without going through the MPU. The transferring of data between primary memory and disk storage is a typical DMA operation.

**HLDA** (Hold Acknowledge): An active high output status signal that indicates to peripheral devices that a *HOLD* request has been received and that the MPU will relinquish the bus on the next clock cycle.

**READY**: This signal is used to synchronize the MPU with slower memory or peripheral I/O devices. During a read or a write cycle, a slow memory or peripheral device can pull the *READY* line low. When this happens, the MPU will wait, or hold, until the *READY* line goes back high. When the slow peripheral is "ready" to complete the data transfer, it raises the *READY* line back high, allowing the MPU to complete the read or write cycle.

**SID** (Serial Input Data): This line is used to input serial data into the accumulator. It will be discussed in detail in Unit 4.

**SOD** (Serial Output Data): This line is used to output serial data from the accumulator. It will also be discussed in Unit 4.

**X1 and X2**: These lines must be connected to an external crystal to establish the internal clock frequency. The clock frequency is the crystal frequency divided by two. Thus, you must use a 4 MHz crystal to operate the 8085 system at 2 MHz.

**CLK OUT** (Clock Output): This line outputs the 8085 clock signal and is used as a system clock line by external devices.

$V_{cc}$ : +5 V power supply

$V_{ss}$ : Ground.

## The 8085 Interrupts

Interrupts are an important part of any system interfacing. An interrupt is an externally generated signal that causes the MPU to deviate from its normal processing sequence in order to serve the interrupting device. The 8085 has several interrupt features, which will be previewed here. You will learn more about them as you do the interfacing and application experiments that follow.

## THE *RESET* INTERRUPT

*RESET IN*: This is the highest priority interrupt. When *RESET IN* is pulled low, the program counter is cleared to zero, the address, data and control buses are tri-stated, and the MPU is reset. The *RESET IN* interrupt is used to re-initialize the system.

*RESET OUT*: This is an active high output status signal that indicates to external devices that the MPU is being reset. This line is used as a system reset line by connecting it to the reset input lines on external devices. This allows peripheral devices to be reset automatically when the MPU resets.

## THE *INTR* (INTERRUPT REQUEST) INTERRUPT

The *INTR* interrupt is a maskable interrupt. A maskable interrupt is one that can be enabled and disabled under software control. When enabled, the interrupt is accepted by the MPU. When disabled, the interrupt is ignored by the MPU. Thus, maskable interrupts provide for software control of the acceptance or rejection of an interrupt.

The *INTR* maskable interrupt is enabled with an EI (Enable Interrupt) instruction and disabled, or masked, using a DI (Disable Interrupt) instruction. If the interrupt is enabled, and the *INTR* line is active (high), the MPU completes the current instruction, accepts the interrupt, and generates an active low output signal called *INTA* (Interrupt Acknowledge).

The *INTA* signal is a response to the *INTR* interrupt input signal. It is used to generate a restart (RST) instruction to the MPU via external hardware. The RST instruction is used to initiate a software routine that will service the interrupt. Such a routine is called an *interrupt service routine*, for obvious reasons.

Here's the idea: There are eight restart (RST) instructions that are recognized by the 8085 MPU. These instructions are listed in Table 1-2, along with their binary and hex representations. Notice that, in each case, the RST instruction is a one byte instruction. In addition, notice that each RST instruction has a specific *call location*.

Table 1-2

The 8085 RST (Restart) Instruction Logic and Call Locations.

INSTRUCTION	BINARY DATA								HEX CODE	HEX CALL LOCATION
	D <sub>7</sub>	D <sub>6</sub>	D <sub>5</sub>	D <sub>4</sub>	D <sub>3</sub>	D <sub>2</sub>	D <sub>1</sub>	D <sub>0</sub>		
RST 0	1	1	0	0	0	1	1	1	C7	0000
RST 1	1	1	0	0	1	1	1	1	CF	0008
RST 2	1	1	0	1	0	1	1	1	D7	0010
RST 3	1	1	0	1	1	1	1	1	DF	0018
RST 4	1	1	1	0	0	1	1	1	E7	0020
RST 5	1	1	1	0	1	1	1	1	EF	0028
RST 6	1	1	1	1	0	1	1	1	F7	0030
RST 7	1	1	1	1	1	1	1	1	FF	0038

The call location is the location in memory where execution is transferred when a given RST instruction is executed. For instance, execution is transferred to memory location 0010<sub>16</sub> when the RST2 instruction is executed.

When a RST instruction is executed, the program counter is stacked and execution is transferred to the call location. A short interrupt service routine or JUMP instruction can be inserted at this location to service the interrupt. Of course, a JUMP instruction would cause the MPU to jump to a service routine located elsewhere in memory.

In any case, a return (RET) instruction must conclude the service routine to allow the MPU to return to where it left off when it was interrupted, so that program execution can continue after the interrupt. The RET instruction simply restores the program counter to its original value by popping the previously saved program counter value off the stack.

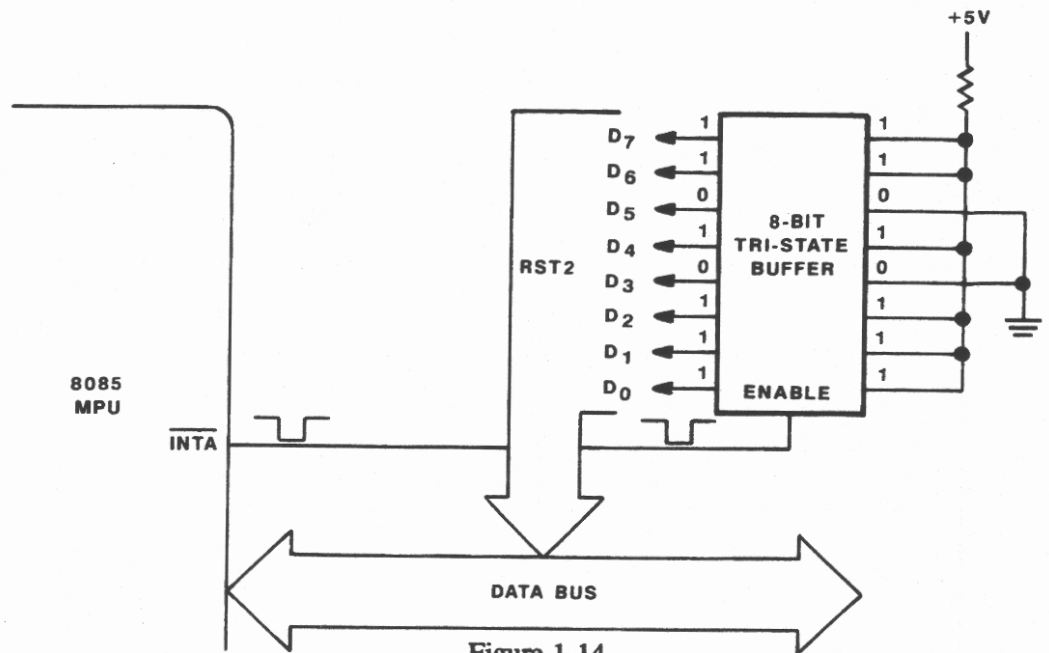


Figure 1-14  
Implementing the RST2 instruction via an external circuit.

Earlier we said that when an *INTR* interrupt is accepted, a RST instruction must be generated *externally*. How can this happen? Well, look at Figure 1-14. Here you see an 8-bit tri-state buffer whose input is *hardwired* (physically wired directly) to generate the binary logic required for a RST2 instruction. The buffer output lines are connected to the MPU data bus. In addition, notice that the latch is enabled by the  $\overline{INTA}$  output signal from the MPU. Recall that when the MPU receives an *INTR* interrupt request, it activates its  $\overline{INTA}$  output signal. From Figure 1-14, you see that this enables the 3-state buffer to generate the RST2 logic onto the MPU data bus. That's all there is to it! The flowchart in Figure 1-15 summarizes the interrupt request process.

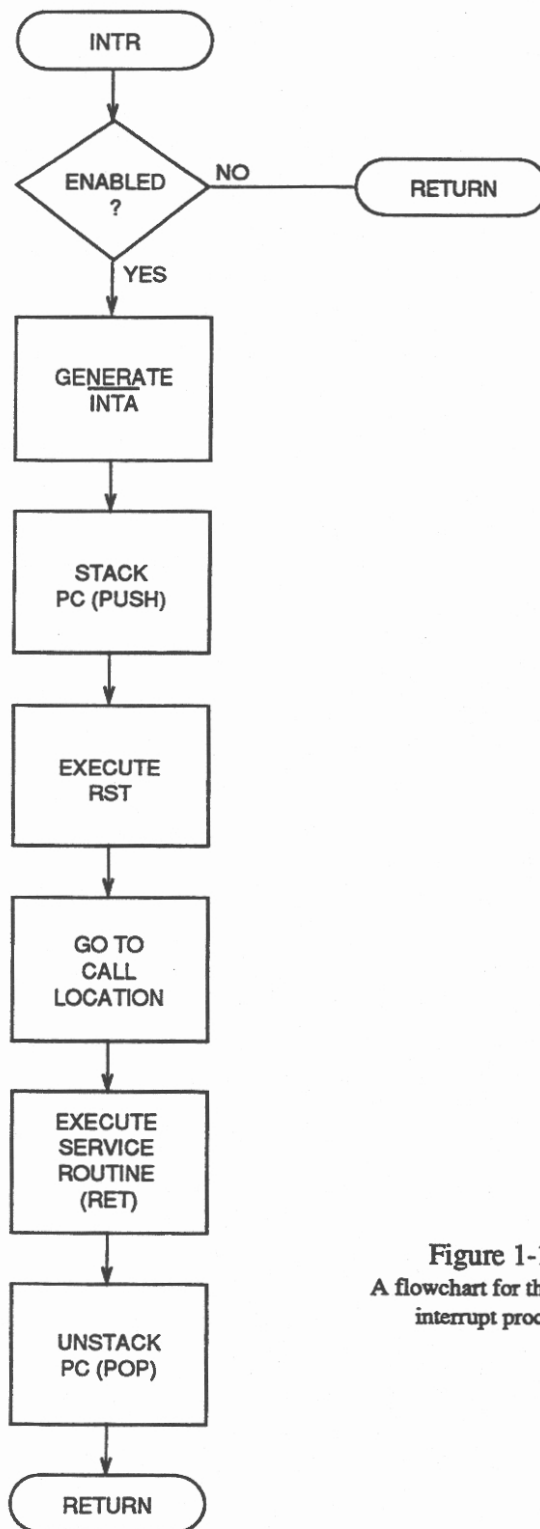


Figure 1-15  
A flowchart for the INTR  
interrupt process.



## THE *TRAP*, *RST 5.5*, *RST 6.5*, AND *RST 7.5* INTERRUPTS

These four remaining 8085 interrupts work a little bit differently than the *INTR* maskable interrupt. They don't require an external circuit to generate a *RST* instruction. Rather, they cause the MPU to automatically vector (go to) to a given address in memory without any external hardware. Table 1-3 shows their relative priority, as well as their call addresses and active states.

Table 1-3  
8085 Interrupt Priorities and Call Locations

INTERRUPT	PRIORITY	CALL LOCATION	TRIGGERING
<i>RESET</i>	1	0000	High edge
<i>TRAP</i>	2	0024	High edge/level
<i>RST 7.5</i>	3	003C	High edge
<i>RST 6.5</i>	4	0034	High level
<i>RST 5.5</i>	5	002C	High level
<i>INTR</i>	6	Depends on <i>RST</i> Instruction	High level

*TRAP* is a non-maskable interrupt that is activated by a rising edge/level. This means that the *TRAP* input line must go high, and stay high, to be accepted. Once accepted, the program counter is stacked and the MPU vectors to address 0024<sub>16</sub>. A short interrupt service routine or *JUMP* instruction is inserted at this address to service the interrupt. An *RET* instruction at the end of the interrupt service routine will pull the program counter from the stack and allow the MPU to begin execution where it left-off before the interrupt.

*RST 5.5*, *RST 6.5*, and *RST 7.5* are all maskable interrupts. Like the *INTR* interrupt, they all can be enabled and disabled using the *EI* and *DI* instructions. In addition, they can be selectively enabled/disabled using a *SIM* (Set Interrupt Mask) instruction. Here's how — The *SIM* instruction is a one byte instruction that reads the 8085 accumulator and enables/disables the interrupts according to the accumulator contents at the time it was read.

The bit map diagram in Figure 1-16 shows how the accumulator bits control the interrupt masking. First, we are only concerned about bits 0 through 4 for this operation. Bit 3 is called the MSE, or Mask Set Enable, bit. This bit must be set for the other bits to have any control over the interrupts. When bit 3 is set, bits 0, 1, and 2 control the masking of *RST 5.5*, *RST 6.5*, and *RST 7.5*, respectively. If a given bit is cleared, the respective interrupt is enabled. Conversely, if a given bit is set, the respective interrupt is disabled, or masked.

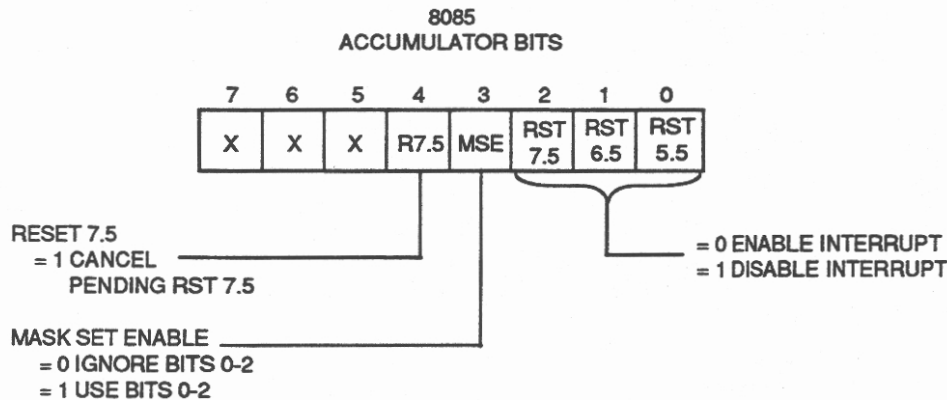


Figure 1-16

Enabling/disabling the *RST* interrupts via the accumulator bits  
and the *SIM* instruction.

As an example, suppose that you want to enable the *RST 5.5* and *RST 7.5* interrupts and disable the *RST 6.5* interrupt. Here's an assembler program that will do the job.

```
MVI A, 0AH
SIM
```

The *MVI* instruction moves the value 0A hex into the accumulator. This value sets the *MSE* and *RST 6.5* bits, while it clears the *RST 5.5* and *RST 7.5* bits. The *MSE* bit must be set for the other bits to have any control over the interrupts. The *RST 6.5* bit is set to disable this interrupt. The *RST 5.5* and *RST 7.5* bits are cleared to enable these interrupts. Once the accumulator bits are set and cleared as required, the one byte *SIM* instruction is executed to actually enable/disable the interrupts.

Notice from Figure 1-16 that bit 4 in the accumulator also relates to the *RST 7.5* interrupt. This bit is used to cancel a pending *RST 7.5* interrupt without servicing it. Setting this bit in the accumulator, then executing a *SIM* instruction will cancel a pending *RST 7.5* interrupt.

There is another one byte instruction called RIM (Read Interrupt Mask) that is associated with the RST interrupts. The RIM instruction is used to determine the status of the RST interrupts. When the RIM instruction is executed, the accumulator is loaded with a value that indicates interrupt status according to Figure 1-17. The lower three bits show the enable/disable status of the three RST interrupts. If a given bit is set, the respective interrupt is masked. The IE bit (Bit 3) shows the enable status of all the interrupts. If IE is set, all the 8085 maskable interrupts, including *INTR*, are enabled. Bits 4, 5, and 6 show the status of the *RST 5.5*, *RST 6.6*, and *RST 7.7*, respectively. If a given bit is set, the respective interrupt is pending.

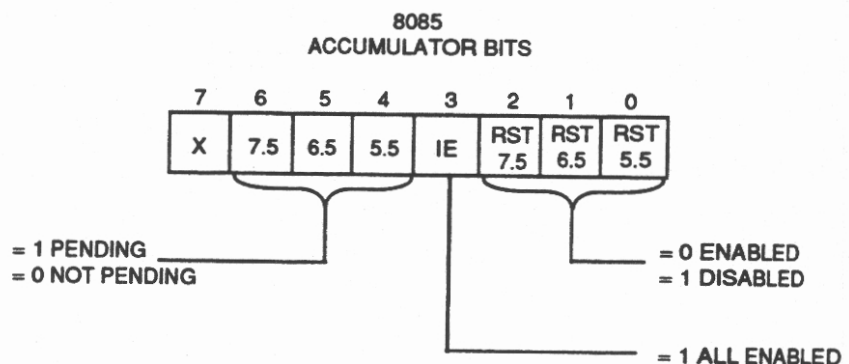


Figure 1-17

Determining interrupt status via the accumulator bits  
and the RIM instruction.

The RIM instruction can be used when servicing a given interrupt to see if another interrupt is pending. For instance, suppose the MPU is currently servicing the *RST 6.5* interrupt and needs to determine if an *RST 5.5* interrupt is pending. Here's a subprogram that will do it:

```
RIM
ANI 10H
JNZ RST6
```

This instruction sequence first uses a RIM instruction to load the accumulator with the interrupt status information. An AND immediate instruction is then executed to check the accumulator bit 4 status. Notice from Figure 1-17 that bit 4 shows the *RST 5.5* pending status. ANDing a logic 1 with this bit will result in a 0 if the bit is cleared and a 1 if the bit is set. If the bit is set, an *RST 5.5* interrupt is pending, the ANDing operation will produce a non-zero result, and the MPU will jump to the RST6 subroutine. Clearly, the routine at the RST6 label would be written to service the *RST 6.5* interrupt.

The last thing you need to know about the RST interrupts is how they are triggered. The RST 7.5 interrupt is positive-edge triggered. As a result, this interrupt can be triggered with a short positive pulse. The interrupt request is internally latched so that it isn't lost before the MPU can respond to it. Of course, a pending RST 7.5 interrupt can be cancelled by setting bit 4 of the accumulator and executing a SIM instruction.

The RST 5.5 and RST 6.5 interrupts are positive-level sensitive. This means that the interrupting device must hold the respective interrupt input line high until the MPU completes the execution of the current instruction, otherwise the interrupt is lost.

## I/O Execution

Now that you are familiar with the 8085 I/O lines, you need to learn how the 8085 performs I/O operations. There are basically two ways that the 8085 MPU can communicate with external devices. The first method is called *direct I/O* and involves special input and output instructions. The second method is called *memory-mapped I/O* and treats I/O devices as if they were memory locations.

Both of these I/O techniques require control signals that are not directly available on the 8085. Direct I/O requires the IOR (I/O Read) and IOW (I/O Write) control signals. Memory mapped I/O requires the MEMR (Memory Read) and MEMW (Memory Write) control signals. These signals are easily obtained from the 8085 IO/M, RD, and WR lines using the external logic shown in Figure 1-18. From now on we will refer to these control signals as if they were available directly on the 8085 chip. Of course, this means that the external logic shown in Figure 1-18 must be in place.

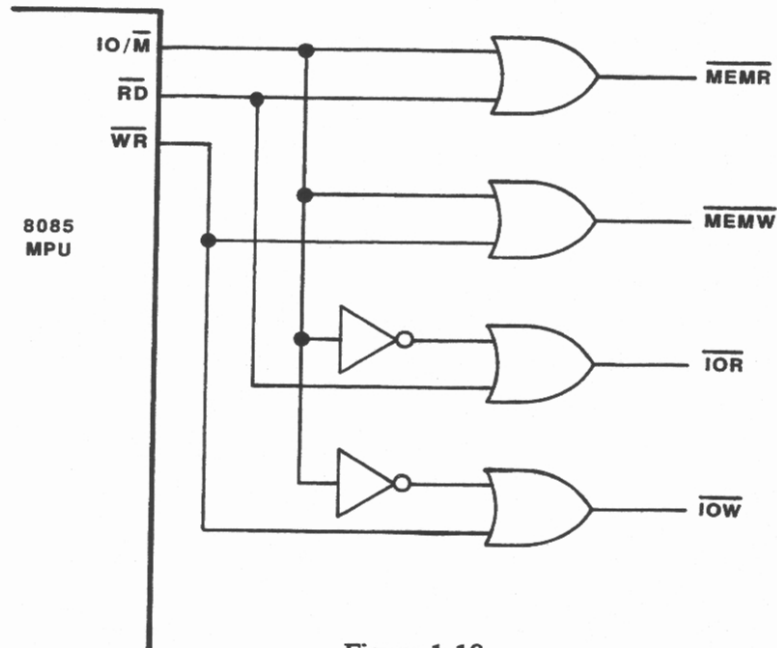


Figure 1-18

Generating the required I/O control signals for direct and memory-mapped I/O.

## DIRECT I/O

Direct I/O is accomplished using two special instructions called IN and OUT. The IN instruction reads data from an input device into the accumulator (for this reason, direct I/O is sometimes referred to as *accumulator I/O*), while an OUT instruction writes the accumulator contents to an output device. When either instruction is executed, the MPU places an 8-bit *port address* on its low address bus. This address can range from 00 through FF hex. Thus, with direct I/O there can be up to 256 I/O ports whose addresses range from 00 to FF hex.

For example, the instruction sequence

```
MVI A, FFH
OUT 01H
```

will output the value FF to port address 01.

On the other hand the instruction

```
IN 02H
```

will input the value from port address 02 into the accumulator.

In order to access a given port, you must decode the lower address bus as shown in Figure 1-19 to generate a port address signal. Of course, the lower address bus must be demultiplexed from the data bus as discussed earlier. Furthermore, the decoder must be designed to respond to the required port address. The output of the decoder must then be NORed with the  $\overline{IOR}$  or  $\overline{IOW}$  control signal to produce an *Enable* signal. The  $\overline{IOR}$  control signal is used for an input port, while the  $\overline{IOW}$  signal is employed for an output port. The output of the NOR gate is used to enable a 3-state buffer to produce an input port or a latch to produce an output port. That's all there is to it!

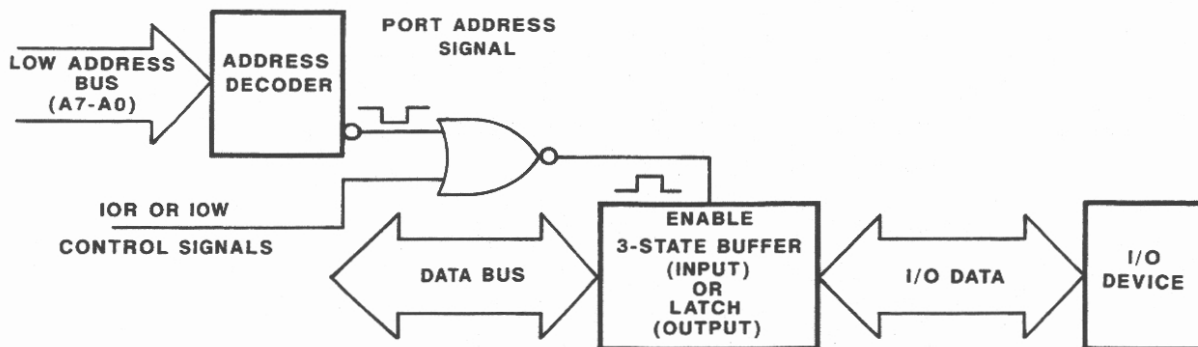


Figure 1-19  
An interface for direct I/O.

## MEMORY-MAPPED I/O

The memory-mapped I/O technique does not employ the IN or OUT instructions. Rather, memory-mapped I/O uses any of the 8085 memory transfer operations such as LDA, STA, LDAX, STAX, and MOV. The reason these instructions can be used is that the I/O device is treated just like a memory location. The MPU simply reads and writes to the I/O locations just as if it were reading and writing to memory.

For instance, the instruction

STA 5000H

would output the accumulator contents to an output device located at address 5000.

On the other hand, the instruction

LDA 6000H

would read the value from an input device located at address 6000 into the accumulator.

How about this one?

LXI H,6000H

MOV A,M

These two instructions would do the same thing as the previous LDA instruction, right? The LXI instruction loads the HL register pair with the input device address (6000). Then, the MOV instruction moves the value at this address into the accumulator.

In order to use memory-mapped I/O, the I/O device must be assigned to an address within the system memory map. Recall that the 8085 can generate 65,536 unique addresses from 0000 through FFFF hex. Once an address is chosen that is not assigned to anything else in the system, an address decoder must be designed to decode the selected address as shown in Figure 1-20. Here, the entire address bus is either completely or partially decoded to generate a device address signal. This signal must then be NORed with the  $\overline{MEMR}$  or  $\overline{MEMW}$  control signal to produce a 3-state buffer or latch enable signal. The  $\overline{MEMR}$  signal is NORed and a 3-state buffer is used for an input device. The  $\overline{MEMW}$  signal is NORed and a latch is employed for an output device.

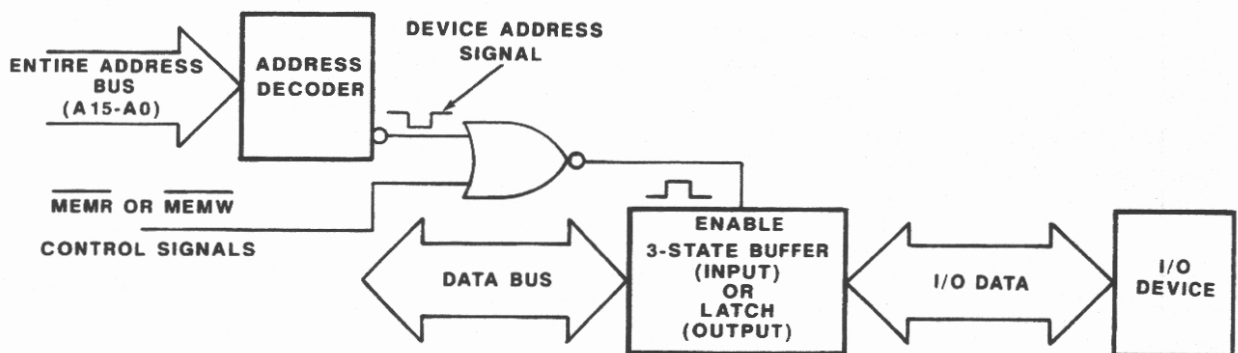


Figure 1-20

An interface for memory-mapped I/O.

There are advantages and disadvantages to both types of I/O just discussed. With direct I/O, only the lower eight address lines need to be decoded. In addition, the entire 64K memory map space is available for memory interfacing and doesn't have to be shared with I/O devices. However, special input and output instructions must be employed which can only operate on the accumulator.

With memory-mapped I/O, the 64K memory space must be shared between memory and I/O devices. The MPU sees these I/O devices as memory locations and, therefore, can employ any of its memory-reference instructions to communicate with the I/O devices. This allows arithmetic and logic operations to be performed directly on I/O data, since instructions such as ADD M, SUB M, and ANA M are available. However, the entire address bus must be completely or partially decoded in order to access a given I/O device.

Table 1-4 provides a comparison between the direct and memory-mapped I/O techniques.

Table 1-4  
Direct versus Memory-Mapped I/O

FACTORS	DIRECT I/O	MEMORY-MAPPED I/O
Control Signals Required	$\overline{\text{IOR}}$ or $\overline{\text{IOW}}$	$\overline{\text{MEMR}}$ or $\overline{\text{MEMW}}$
Address Decoding	Lower eight address lines	All sixteen address lines
MPU Registers Employed	Accumulator only	Any register
Instructions	IN and OUT only	Any memory-reference instruction
I/O Ports	256 input and 256 output	Any number shared with memory



## Self-Test Review

8. Into what three functional categories can the 8085 I/O lines be divided?
9. Why and how must you demultiplex the low order address bus?
10. Which 8085 I/O line is used when interfacing with slow peripherals?
11. What is the function of the SID and SOD I/O lines?
12. Explain how the *INTR* interrupt works.
13. What are the 8085 non-maskable interrupts?
14. How are the RST interrupts masked?
15. Explain direct I/O.
16. Explain memory-mapped I/O.

## Answers

8. Data, address, and control/status.
9. The low order address bus must be demultiplexed because it is combined with the data bus on I/O lines  $AD_0$  through  $AD_7$ . To demultiplex it, you must use an external 8-bit latch and enable the latch using the *ALE* control line (See Figure 1-13).
10. READY
11. For serial I/O in and out of the accumulator.
12. When the *INTR* interrupt is accepted, the 8085 generates an interrupt acknowledge signal on its *INTA* line. This signal is used to enable an external hardwired 3-state buffer to generate a RST instruction onto the data bus. The RST instruction causes the MPU to go to a call address where a short service routine or JUMP instruction is located. A RET instruction at the end of the interrupt service routine causes the MPU to return to where it left-off when it was interrupted (See Figure 1-15).
13. RESET and TRAP.
14. Using a SIM instruction.
15. Direct I/O employs the IN and OUT instructions to input and output data, respectively, via the accumulator. The lower address bus is decoded along with either the *IOR* or *IOW* control line to form a port address that ranges from 00 to FF hex. There can be 256 input and 256 output port addresses that are separate from the 8085 memory map space.
16. Memory-mapped I/O requires that the I/O device share the memory space with the system memory. As a result, all 16 address lines are decoded to map the I/O devices within unused areas of the memory space. With memory-mapped I/O, any of the 8085 memory-reference instructions can be employed to communicate with I/O devices.

## INTERFACING WITH SWITCHES — DATA INPUT

The most popular input device used with the microprocessor is the switch. The operator of microprocessor-based equipment usually communicates with the MPU by pushbutton switches, limit switches, pressure switches, etc. In this section, you will examine some techniques used in interfacing with switches.

### Interfacing Requirements

When interfacing with a switch, or switches, four operations are involved. First, the MPU must select or address the proper switch bank. Second, it must detect contact closure. Third, it must provide for debouncing (unless this is accomplished by external hardware). Finally, it must decode the input. The following is a description of each of these operations in more detail.

### SELECTING THE SWITCH

Figure 1-21 shows a simple arrangement for connecting eight switches to the MPU. Three-state buffers are used to interface the switches with the data bus. The buffers are enabled by the output of the address decoder. This decoder can use any of the decoding schemes discussed earlier. Let's assume that the decoder responds to address  $C003_{16}$ . Until the decoder receives this address, the buffers are disabled. This effectively disconnects the switches from the data bus.

To find out if a switch is closed, the MPU must read in the data from this address. An easy way to do this is with the LDA instruction. When the LDA  $C003H$  instruction is executed, the address  $C003$  goes out on the address bus. The decoder detects this address and enables the three-state buffer. Thus, for an instant, the switch bank is connected to the data bus. The data from the switch bank is loaded into the accumulator.

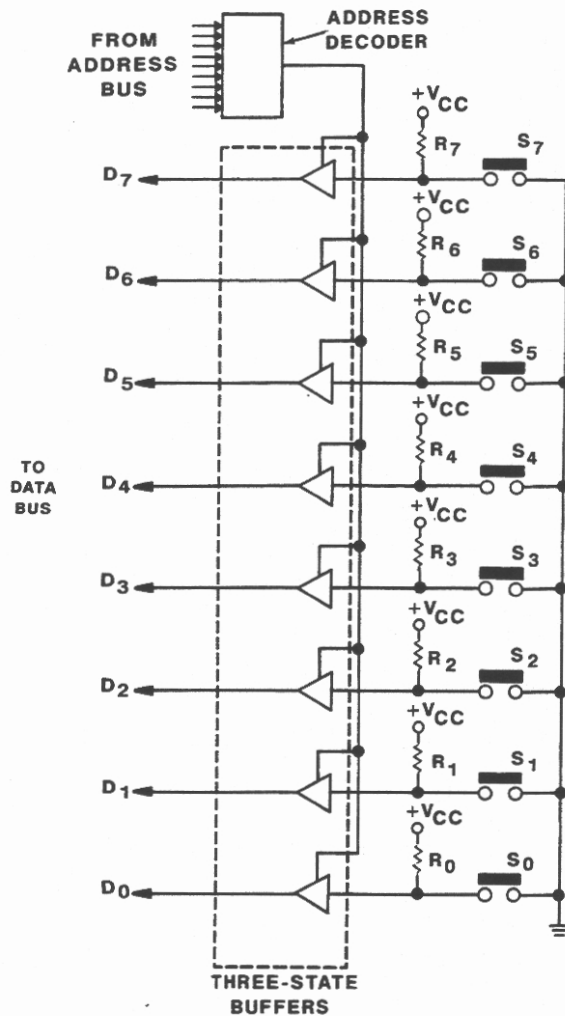


Figure 1-21

Interfacing a bank of switches to the MPU.

## DETECTING CONTACT CLOSURE

If none of the switches are closed, all the data lines will be high because of pull-up resistors  $R_0$  through  $R_7$ . Thus, the data entered into the accumulator will be  $FF_{16}$ . To test for a switch closure, the contents of the accumulator can be compared with  $FF_{16}$ . That is, a CPI  $FFH$  instruction could be used. If this is followed by a JNZ instruction, the MPU will jump if a key is depressed. Otherwise, it will not. For example, suppose  $S_0$  is closed. When the accumulator is loaded from address  $C003_{16}$ , the  $D_0$  line will be low. Thus, the number loaded into the accumulator will be  $FE_{16}$ . The CPI instruction clears the zero (Z) flag since no match occurs. Therefore, the JNZ instruction causes the branch to occur.

## DEBOUNCING THE SWITCH

Most mechanical switches produce contact bounce. When the switch is closed, the contacts do not make an immediately solid electrical or mechanical connection. Instead, they "bounce" open and closed for a brief period of time. Figure 1-22 illustrates this effect.

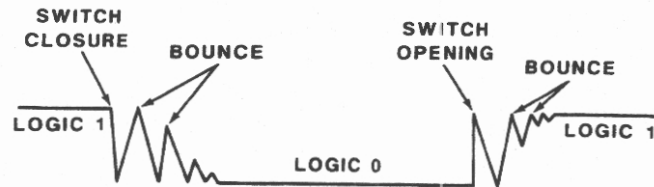


Figure 1-22  
The effects of contact bounce.

While contact bounce may only last for a few milliseconds, this is long enough for the MPU to interpret the bounce as repeated switch closures. To overcome this bounce problem, some switches use cross-coupled NAND gates that will immediately latch in one state so that all contact bounce is ignored. However, this requires additional circuitry.

In many applications, a better approach is to let the MPU itself do the debouncing. A simple scheme is to wait about ten milliseconds and then read in the data from the switch bank again. If the same indication occurs, then the MPU can be certain that the switch is closed. The switch can be checked as many times as is necessary to ensure that contact bounce is eliminated.

## DECODING THE SWITCHES

After the MPU determines that a switch has been closed, it must decide which switch it is. In most cases the switch closure represents a number. For example, the MPU should recognize an S5 closure as the number 5. This, too, is easily accomplished by the proper subroutine.

Referring to Figure 1-21, you can see that each switch corresponds to one bit of the data line. When a switch is closed, the corresponding data line goes to 0. When loaded into the accumulator, the corresponding bit is also 0. The bit that is 0 can be detected by rotating the accumulator into the carry bit until the carry bit is cleared. By counting the number of rotations, the MPU can determine which switch is depressed.

In most applications, another job of the decoding procedure is to reject multiple switch closures. If two switches are closed simultaneously, the MPU should not accept data. If a second switch is closed before the first switch is released, the

MPU may reject the data or accept only the first switch closure. By using a few extra programming steps, a very simple and inexpensive keyboard can appear quite sophisticated.

## A Typical Keyboard Arrangement

A simplified keyboard arrangement is shown in Figure 1-23.

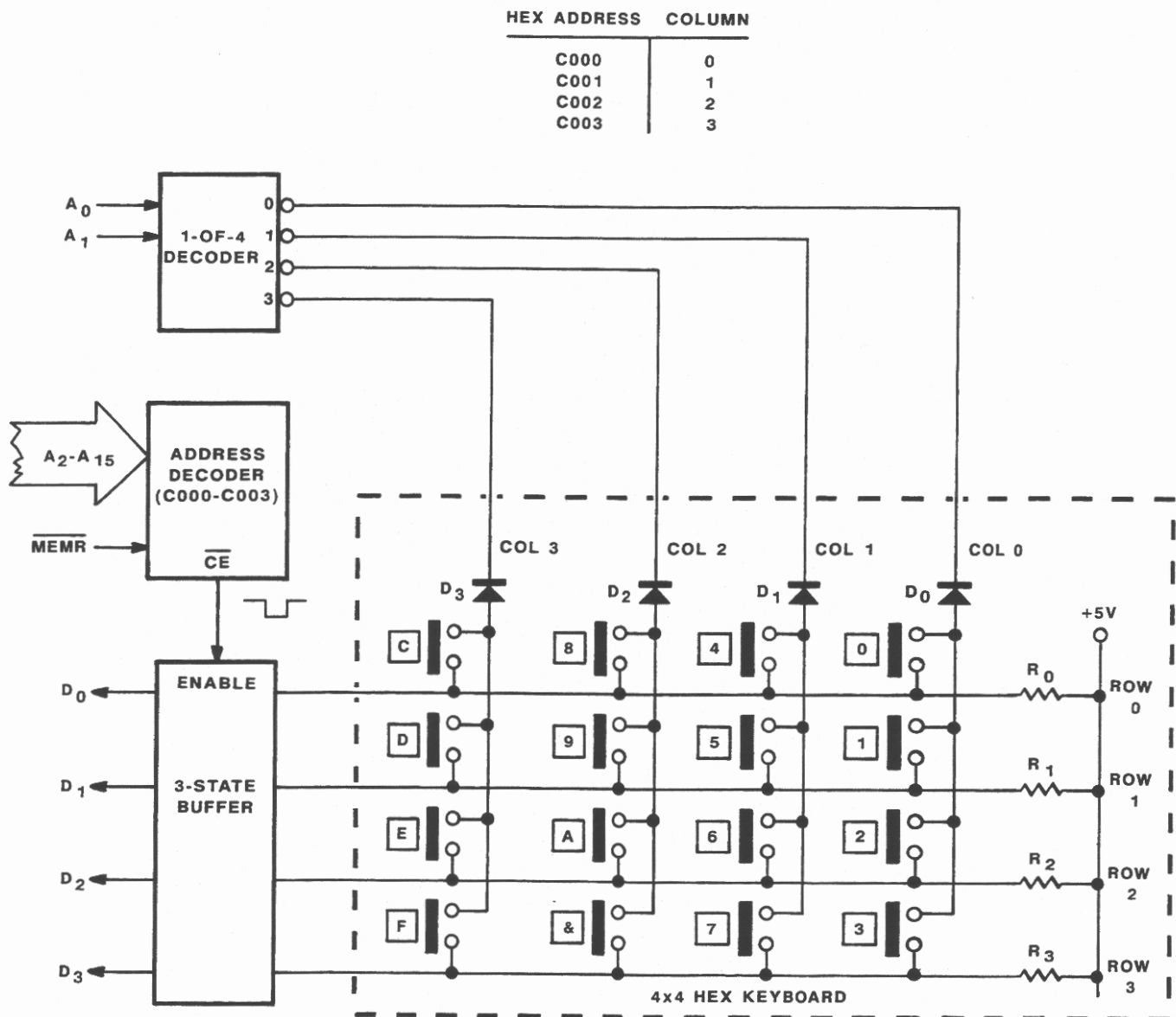


Figure 1-23  
A typical keyboard arrangement.

## THE CIRCUIT

As you can see, the circuit consists of an address decoder, a 3-state buffer, a keyboard column decoder, and a  $4 \times 4$  hex keyboard. You are already familiar with the function of the address decoder and 3-state buffer. Notice that the address decoder is decoding the upper fourteen address lines ( $A_2 - A_{15}$ ) along with the MPU  $\overline{MEMR}$  control line. The decoder is designed to respond to hex addresses C000 through C003. When an address within this range appears on the address bus, the decoder will enable the 3-state buffer. Notice that the 3-state buffer is a 4-bit buffer that connects the keyboard rows to the lower four data bus lines ( $D_0 - D_3$ ).

The hex keyboard is constructed by placing sixteen pushbutton switches in a  $4 \times 4$  array. The rows of the array are connected on the left side to data bus lines  $D_0 - D_3$  via the 3-state buffer. The rows are connected on the right to the +5V supply potential through four pull-up resistors ( $R_0 - R_3$ ). The columns of the array are connected, via four diodes, to the output of a simple 1-of-4 decoder which is being driven by address lines  $A_0$  and  $A_1$ . Notice that the sixteen pushbutton switches are located at the intersections of the array rows and columns. In fact, when a given pushbutton is depressed, it makes a connection between its respective row and column.

Here's how it all works. When a decoded address appears on the address bus, the three state buffer is enabled and a logic 0 is placed at the top of one of the four keyboard columns. For instance, suppose the MPU places address C003 on the address bus. It is obvious that this address enables the 3-state buffer. In addition, output line 3 of the keyboard column decoder generates a logic 0 on column 3 of the keyboard. This is because address lines  $A_0$  and  $A_1$  are both at a logic 1 state (11) for address C003. The other output lines of the column decoder are inactive (logic 1). The logic 0 output of the column decoder is applied to the cathode of the column 3 diode. As a result, the diode is forward biased and acts like a closed switch. This applies a logic 0 on the entire column 3 line.

Now, suppose all the pushbutton keys in column 3 are open. When this is the case, all the rows are at a logic 1 state via the pull-up resistor arrangement. However, suppose that you depress one of the column 3 keys. For example, suppose that the 'C' key is depressed. This connects column 3 to row 0. Since a logic 0 is being applied to column 3, this logic 0 state is passed through the key to row 0. As a result, the  $D_0$  data line will be at a logic 0 state rather than its normal logic 1 state.

So, the idea is to generate an address to the keyboard circuit that will apply a logic 0 to a given column. If any of the keys within that column are depressed, the logic 0 state is passed on to the respective data bus line via the depressed key and the 3-state buffer. The table in Figure 1-20 shows the address versus column selection arrangement for this circuit.

Next, as with the column switch arrangement you observed earlier, there are three major software tasks to be performed when working with a keyboard array: detect a key closure, debounce the closure, and decode the key closure. We will develop three subroutines to perform these required tasks, beginning with detecting the key closure.

## DETECTING KEY CLOSURE

In order to detect key closure, the MPU must address the keyboard and simply verify that there is a logic 0 on one of the four data bus lines,  $D_0$  through  $D_3$ . If none of the keys are depressed all the data lines will be at a logic 1 state. However, if any of the four keys in the addressed column are depressed, one of the data lines will be at a logic 0 state.

In this course, we will first write a "generic" algorithm for our software routines instead of jumping right into the micro code. You will find it much easier to understand the software process and to code in assembly language once you develop a good algorithm. Here's an algorithm to perform the key closure detection task.

### SUBROUTINE KEYCHECK

BEGIN

Load beginning keyboard address into index pointer register.

REPEAT Load accumulator indirect via index pointer.

Increment index pointer.

Mask accumulator data bits  $D_4 - D_7$ .

Compare 1's to accumulator data bits  $D_0 - D_3$ .

Jump zero to REPEAT.

Return

END.



The algorithm begins by loading the beginning keyboard address (C000 in our case) into an MPU index register pair. The accumulator is then loaded indirect, via this index pointer register. Since our beginning keyboard address is C000, the accumulator will be loaded with the data at this address. What's at address C000? You're right, the Column 0 switch data from the keyboard.

The index pointer register is then incremented to point to the next keyboard address. This address will be loaded shortly. However, before it is, the MPU must determine that there are no logic 0's in accumulator bits  $D_0 - D_3$  from Column 0. To do this, the unwanted upper four data bits ( $D_4 - D_7$ ) are masked. Then, a compare operation is performed to check the  $D_0 - D_3$  bit status. Recall that a compare operation is actually a subtract operation. If you subtract 1's from the  $D_0 - D_3$  bits and obtain a zero result, you know that all the data bits must be set (logic 1). If one of the bits is cleared (logic 0), the result of the compare operation is not zero and the MPU has detected a key closure. If a closure is detected the MPU falls out of the subroutine and returns.

Notice that if the compare operation generates a zero result, none of the accumulator bits are logic 0 and the subroutine jumps to REPEAT. Here, the accumulator is loaded with the next sequential column address, since remember that the index pointer register has been previously incremented to point to this next column address. After the accumulator is loaded, the pointer register is incremented again to point to the next column address. But before this column is loaded, the MPU tests its lower four data bits for a key closure as before. If no key closure is detected, the next column address is loaded, tested, and so on, until a key closure is detected.

In summary, the subroutine is simply scanning the keyboard, column-by-column, until it detects a key closure via a logic 0 present in one of the lower four accumulator bits. Now, by working out a good algorithm, the coding of an 8085 assembly language program is simple. Just follow the algorithm using the available 8085 registers like this:

```
                                PUSH D
                                PUSH PSW
                                LXI D,C000H
REPEAT  LDAX D
                                INR E
                                ANI 0FH
                                CPI 0FH
                                JZ REPEAT
                                POP PSW
                                POP D
                                RET
```

Notice that we have chosen to employ the DE register pair as the index pointer register. Of course, the MPU registers in use must be pushed on the stack at the beginning of the subroutine and popped at the end so that the MPU can return where it left off. Make sure you understand how the body of the above 8085 program performs the algorithm we just discussed.

It is important to note that after a key closure is detected, the MPU only knows that a key has been depressed. It doesn't know which key has been depressed at this point. This is the job of the decoding subroutine, but first we must debounce the key closure.

## DEBOUNCING KEY CLOSURE

To debounce the key closure, the MPU must simply delay about 10 milliseconds. Here is a delay algorithm that should do the job:

### SUBROUTINE DEBOUNCE

BEGIN

Load delay count value into MPU register pair.

LOOP       Decrement register pair.

Compare register pair for zero.

Jump if not zero to LOOP.

Return

END.

As you can see, the algorithm simply loads an MPU register pair with a count value that is large enough to create the required delay. The register pair is then decremented down to zero to generate the delay. Here's the 8085 program.

```

                                PUSH H
                                PUSH PSW
                                LXI H,COUNT
LOOP    DCX H
                                MOV A,H
                                ORA L
                                JNZ LOOP
                                POP PSW
                                POP H
                                RET
```

Here, we are using the HL register pair as the delay counter. Do you see how the register pair is decremented and tested for zero? The DCX instruction decrements the register pair, while the MOV, ORA, and JNZ instructions test the decremented register pair for zero.

Once the key closure is debounced, the last task is to decode the closure.

## DECODING KEY CLOSURE

This task requires the MPU to determine exactly which key is depressed. Again, we will discuss the process by way of an algorithm. Here it is:

### SUBROUTINE DECODE

#### BEGIN

Clear key Code register.

Load beginning keyboard address into index pointer register.

Set Column register to 04.

COL      Load accumulator indirect via index pointer.

Increment index pointer.

Set Row register to 04.

ROW      Rotate accumulator right into carry flag.

Jump if C = 0 to DONE.

Increment key Code register.

Decrement Row register.

Jump if not zero to ROW.

Decrement Column register.

Jump if not zero to COL.

DONE      Move code register value to accumulator.

Return.

END.

Here's the general idea of the process. The above algorithm will scan the keyboard, column-by-column, row-by-row, looking for the depressed switch. Remember that the depressed switch will generate a logic zero on its respective data line. The keyboard scan begins with Column 0, Row 0 then proceeds down Column 0, checking all the rows in this column. The scan then jumps to the top of Column 1 and proceeds to check all the rows in this column, and so on, until the depressed key is found. In other words, the scan is from right-to-left, top-to-bottom, in the numeric order of the keys shown in Figure 1-20. As the scan proceeds, a Code register is incremented until the depressed key is found. At this point, the scan stops and the Code register will contain the depressed key number.

Looking closer at the algorithm, you will note that four MPU registers are being employed. Here are the functions of each:

- Code Register: To be incremented during the scan so that it contains the depressed key number when the subroutine is completed.
- Index Pointer Register: To point to the current column address being accessed.
- Column Register: To control the number of keyboard columns to be scanned.
- Row Register: To control the number of keyboard rows to be scanned.
- Accumulator: To hold the key data for the column currently being scanned. Note that the four lower bits within the accumulator represent the four keys rows within the column being scanned.

The main body of the algorithm consists of two loops: an outer column loop (COL) and an inner row loop (ROW). The outer COL loop controls the column scan, while the inner ROW loop control the row scan within a given column.

Here's how it works. The beginning keyboard address (C000 in our case) is loaded into the index pointer register. The column register is then initialized with the value 04, since there are four columns to be scanned. As a result, the COL loop will be executed at most four times.

Next, the accumulator is loaded with the key data from Column 0. At this time, the index pointer register is incremented to point to the next column in preparation for the second pass through the COL loop. The row register is then initialized with the value 04, since there are four rows to be scanned. Thus, the ROW loop will be executed at most four times.

At this point, the accumulator contains the logic generated by keys 0 through 3 in its lower four data bits. Remember that the MPU must detect a logic 0 in one of these bits to detect a key closure. To do this, the MPU rotates the four bits one at a time through the carry flag within the ROW loop. If a logic 0 is found in the C flag, the key closure has been detected and the subroutine is exited. Notice that after each rotate and test operation, the Code register is incremented to indicate the numeric value of the key to be tested next. This is so the Code register will contain the numeric value of the depressed key when the subroutine is exited.

The ROW loop is executed at most four times to check each of the four data bits in Column 0. If none of the bits are a logic 0, the column register is decremented, and the subroutine jumps to the outer COL loop. This time, the accumulator is loaded with the data from Column 1, the row register is re-initialized with the value 04 and the ROW loop is executed at most four more times to check the data in this column. The process is then repeated for Columns 2 and 3, until the depressed key is detected. That's all there is to it! When the key closure is decoded, the key value contained in the Code register is placed in the accumulator and the subroutine returns to the calling program. Now, following the above algorithm the 8085 assembly language program is:

```

                                PUSH B
                                PUSH D
                                PUSH H
                                LXI H,0000H
                                LXI D,C000H
                                MVI C,04H
COL      LDAX D
                                INR E
                                MVI B,04H
ROW      RAR
                                JNC DONE
                                INR L
                                DCR B
                                JNZ ROW
                                DCR C
                                JNZ COL
                                MOV A,L
                                POP H
                                POP D
                                POP B
                                RET

```

Here, we have employed the 8085 registers as follows:

Register L: Code Register

Register Pair DE: Index Pointer Register

Register C: Column Register

Register B: Row Register

Of course, we have pushed these registers onto the stack in the beginning of the subroutine and popped them at the end to restore them to their pre-subroutine values. You might want to compare the above code to the algorithm to make sure you understand how the code implements the algorithm.

This keyboard array could easily be expanded to an  $8 \times 8$  array with 64 keys. The software process would be essentially the same, but utilizing all eight MPU data bus lines.

You will learn more about interfacing with switches later after you learn about a programmable interface device. Also, in a later experiment you will gain some practical experience interfacing switches.

## Self-Test Review

17. List four requirements that must be met when connecting mechanical switches to the MPU.
18. What type of circuit is often used between the switches and the data bus of the MPU?
19. Refer to Figure 1-21. If no switches are closed, what hexadecimal number is read from the switch bank?
20. Refer to Figure 1-21. If  $S_2$  is closed, what hexadecimal number is read from the switch bank?
21. What two things can be determined by the hexadecimal number read in from the switch bank in Figure 1-21?
22. Why is debouncing required?
23. How can the MPU overcome the effects of contact bounce?
24. Refer to Figure 1-23. To what address does the Column 2 of keys respond?
25. How is key closure initially detected in Figure 1-23?
26. What technique is used for finding the hexadecimal equivalent of the key that is depressed in Figure 1-23?
27. How does the DEBOUNCE subroutine overcome contact bounce?

## Answers

17. The MPU must:
  - 1) Address the switches.
  - 2) Detect contact closure.
  - 3) Overcome contact bounce.
  - 4) Decode the switches.
18. A three-state buffer.
19.  $FF_{16}$ .
20.  $FB_{16}$ .
21. The hexadecimal number read from the switch bank reveals
  - 1) If a switch is closed.
  - 2) Which switch is closed.
22. When a mechanical switch is closed, its contacts often bounce open and closed several times. Unless this is taken into consideration, a single switch closure can be mistaken for multiple switch closures.
23. The MPU can overcome the effects of contact bounce by executing a software delay after closure has been detected.
24.  $C002_{16}$ .
25. By scanning the columns, loading column data into the accumulator, masking the upper four bits, and comparing the lower four bits to ones.
26. Scanning column-by-column, row-by-row until a logic 0 is detected in the C flag. A code register is incremented throughout the scan to indicate the key number.
27. By delaying approximately 10 ms after switch closure has been detected.



## INTERFACING WITH DISPLAYS — DATA OUTPUT

The 7-segment LED is a popular device for displaying a single character. It can be used to display the decimal digits 0 through 9 or the hexadecimal digits 0 through F. It can also display many special characters. Its low cost and flexibility make the 7-segment LED display ideal for low cost microprocessor based systems.

Recall that there are two basic types of 7-segment displays. One is called the *common-anode* type. A segment is forward biased (turned on) by applying a low logic level to its cathode. An external series resistor is required to reduce the current to an acceptable level.

The other type is called the *common-cathode* type. In this case, a segment is forward biased (turned on) by applying a high logic level to its anode.

### Driving the 7-Segment Display With the MPU

A number of IC's are especially designed to drive the 7-segment display. A typical example is the 7447 shown in Figure 1-24A. This is a BCD-to-7-segment decoder-driver. It receives a 4-bit binary number at inputs A, B, C, and D. It provides the proper patterns at outputs a through g to form the numerals 0 through 9 and six special characters as shown in Figure 1-24B.

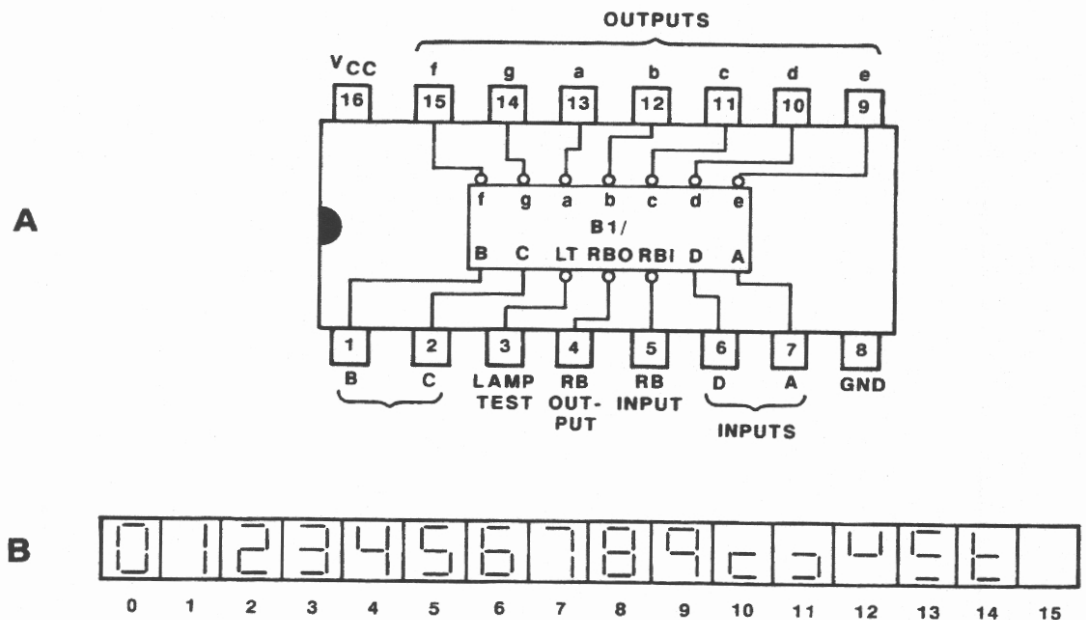


Figure 1-24

The 7447 seven segment decoder driver and its resultant displays.

When used with a microprocessor, the MPU does not drive the decoder-driver directly. Recall that the information on the data bus is there for a microsecond or less. Since the 7447 has no latch capability, a separate latch must be used to store the data at the right instant. Figure 1-25 shows a representative circuit.

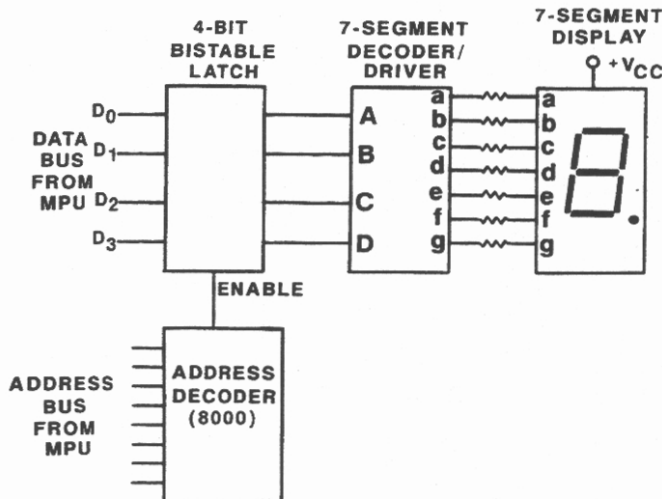


Figure 1-25  
Using the decoder/driver.

Here a 4-bit bistable latch is used between the data bus and the decoder-driver. The latch is enabled only at the instant that its address appears on the address bus. For example, assume that the latch has been given the address  $8000_{16}$ . An instruction such as  $STA\ 8000H$  will activate the display. When this instruction is executed, the address 8000 is placed on the address bus. The address decoder recognizes this address and enables the latch. Thus, the data on lines  $D_0$  through  $D_3$  are latched into the 4-bit latch. An instant later, the MPU places new information on the address and data buses. However, the display responds only to what has been preserved in the 4-bit latch.

The advantage of this circuit is that it requires only one instruction from the MPU to display a given number indefinitely. Its disadvantage is its lack of flexibility. Of the  $256_{10}$  displays possible, only the  $16_{10}$  provided by the decoder-driver can be used. Thus, this arrangement can display only those symbols shown in Figure 1-24B.

A more versatile way of driving a 7-segment display is shown in Figure 1-26. Here, a low current display is used so that the latch can drive the display directly. Notice that the BCD-to-7-segment decoder is eliminated. The seven segments of the display (and the decimal point) are now controlled directly by the eight bits of data on the data bus.

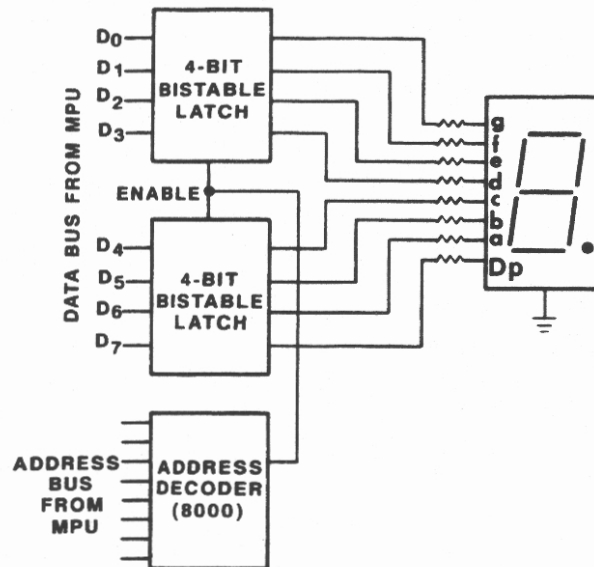


Figure 1-26

A more versatile way of driving a 7-segment display.

The display shown is a common-cathode type. Thus, a segment is turned on by applying a logic high to its input. Figure 1-27 shows the 8-bit pattern that is required to display the digit 1. Since this 8-bit pattern comes from the MPU, the microprocessor must do the decoding. This requires more MPU time and instructions but it greatly increases the versatility of the display. We can now use any of the  $256_{10}$  possible displays by providing the proper 8-bit pattern.

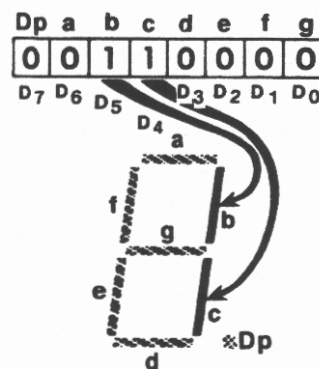


Figure 1-27

Each bit controls one segment of the display.

A display of this type is often used to display the hexadecimal digits 0 through F. The MPU must convert the binary numbers 0000 through 1111 into the proper 8-bit patterns to display the equivalent hexadecimal digit. The easiest way to do this is to use a "look-up" table. The 8-bit patterns are placed in  $16_{10}$  consecutive memory locations. Assume that the starting address of the table is  $FF96_{16}$  as shown in Figure 1-28. In addition, we will assume that the accumulator contains the binary equivalent of the value to be displayed. Here's an algorithm for a subroutine that could be called to display the accumulator value:

#### SUBROUTINE DISPLAY1

BEGIN

Load an index pointer register with the beginning look-up table address.

Add the beginning look-up table address to the display value in the accumulator to form the display code address.

Move the display code address to the index pointer register.

Move the display code into the accumulator from the display code address.

Store the display code to the display.

Return

END.

The subroutine begins by loading an index pointer register with the beginning look-up table address. Next, this address is added to the display value already in the accumulator. The sum, which is the display code address, is then moved to the index pointer register. For instance, suppose the beginning address of the look-up table is  $FF96$  as in Figure 1-28, and the display value is 06. The index pointer is first loaded with the value  $FF96$ . This value is then added to the accumulator value (06) to obtain  $FF9C$ . Notice from Figure 1-28 that  $FF9C$  is the address of the display code for the value 6.

HEX ADDRESS	HEX CONTENTS	MNEMONIC/CONTENTS	COMMENTS
		<u>D<sub>p</sub> a b c d e f g</u>	
FF96	7E	0 1 1 1 1 1 1 0	7-Segment pattern for 0
FF97	30	0 0 1 1 0 0 0 0	7-Segment pattern for 1
FF98	6D	0 1 1 0 1 1 0 1	7-Segment pattern for 2
FF99	79	0 1 1 1 1 0 0 1	7-Segment pattern for 3
FF9A	33	0 0 1 1 0 0 1 1	7-Segment pattern for 4
FF9B	5B	0 1 0 1 1 0 1 1	7-Segment pattern for 5
FF9C	5F	0 1 0 1 1 1 1 1	7-Segment pattern for 6
FF9D	70	0 1 1 1 0 0 0 0	7-Segment pattern for 7
FF9E	7F	0 1 1 1 1 1 1 1	7-Segment pattern for 8
FF9F	7B	0 1 1 1 1 0 1 1	7-Segment pattern for 9
FFA0	77	0 1 1 1 0 1 1 1	7-Segment pattern for A
FFA1	1F	0 0 0 1 1 1 1 1	7-Segment pattern for B
FFA2	4E	0 1 0 0 1 1 1 0	7-Segment pattern for C
FFA3	3D	0 0 1 1 1 1 0 1	7-Segment pattern for D
FFA4	4F	0 1 0 0 1 1 1 1	7-Segment pattern for E
FFA5	47	0 1 0 0 0 1 1 1	7-Segment pattern for F

Figure 1-28

Program segment and table for converting binary  
to common cathode 7-segment display format.

Now that the display code address is in the index pointer register, the code at this address can be moved to the accumulator. Once in the accumulator, the display code is stored to the display and the subroutine returns to the calling program. Here's the 8085 assembler code that will implement the DISPLAY algorithm:

```

PUSH H
LXI H,FF96H
ADD L
MOV L,A
MOV A,M
STAX 8000H
POP H
RET

```

Here, we are using the 8085 HL register pair as the index pointer register. Thus, this register pair is pushed onto the stack at the beginning of the subroutine and popped at the end of the subroutine. As you can see, the HL register pair is first loaded with the beginning address of the look-up table in Figure 1-28. Then, the lower register of the pair (L) is added to the accumulator which contains the binary display value. The sum is the display code address. This sum is then moved back to the L register so that the HL register pair points to the proper display code. The display code at this address is then moved into the accumulator.

Finally, this number is stored at address  $8000_{16}$ . This is the address of the latches that drive the display. Therefore, if  $06_{16}$  is in accumulator A when this program segment is executed, the digit 6 will be displayed. Also, if  $02_{16}$  is initially in accumulator A, the program segment will cause a 2 to be displayed.

Using this arrangement, two 4-bit latches are required for each display. There are eight bit latches available in a single IC. However, most come in 20 to 24-pin packages, which are relatively expensive. There is one type of 8-bit latch that comes in a 16-pin package. This device will be discussed next.

## Using an Addressable Latch

Figure 1-29 shows the pin outs and schematic for the 8-bit addressable latch. It can store an 8-bit byte and drive a low current display. Its most striking characteristic is that data is entered into the device in serial form. That is, it has a single data input (pin 13). Thus, the eight bits of data must be entered into the device one bit at a time. This explains how the IC can get by with only 16 pins.

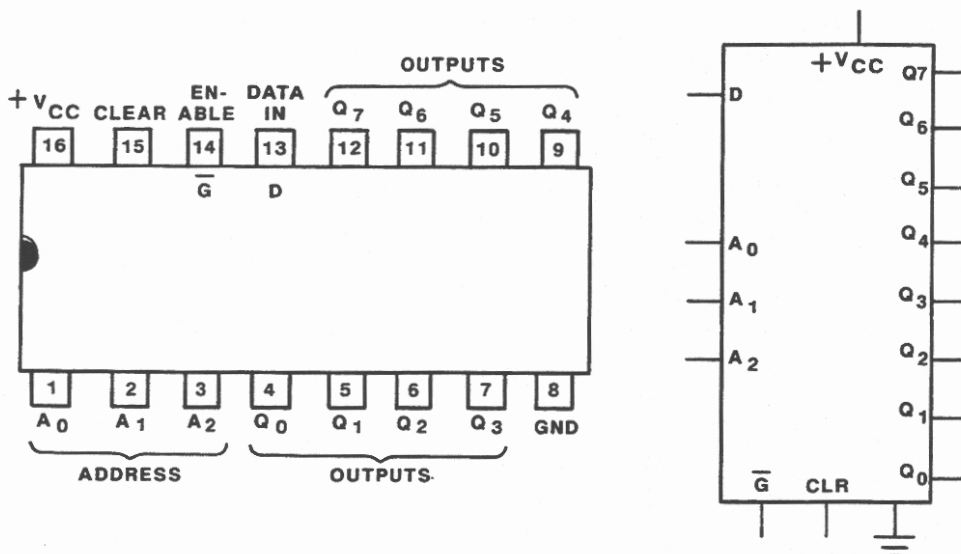


Figure 1-29

Pin outs and schematic diagram of the 74LS259 latch.

The addressable latch has an enable pin that allows it to be selected by an address decoder. A low at pin 14 will enable the latch and allow it to receive data. Actually, there are eight latches on the IC. They are numbered 0 through 7. The particular latch to which the input data bit is routed is determined by the 3-bit address at pins  $A_0$ ,  $A_1$ , and  $A_2$ . Figure 1-30 shows which latch is selected for each address.

ADDRESS INPUTS			LATCH SELECTED
A <sub>2</sub>	A <sub>1</sub>	A <sub>0</sub>	
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

Figure 1-30

Latch selection table.

How can this latch be used by the microprocessor to drive a display? Figure 1-31 shows a representative circuit. The three address lines on the addressable latch are connected to their corresponding lines on the address bus. The remaining lines of the address bus are connected to the address decoder. Assume that the address decoder is arranged so that the addressable latch is enabled for eight different addresses. Address  $C160_{16}$  selects latch 0 of the addressable latch;  $C161_{16}$  selects latch 1; and so forth.

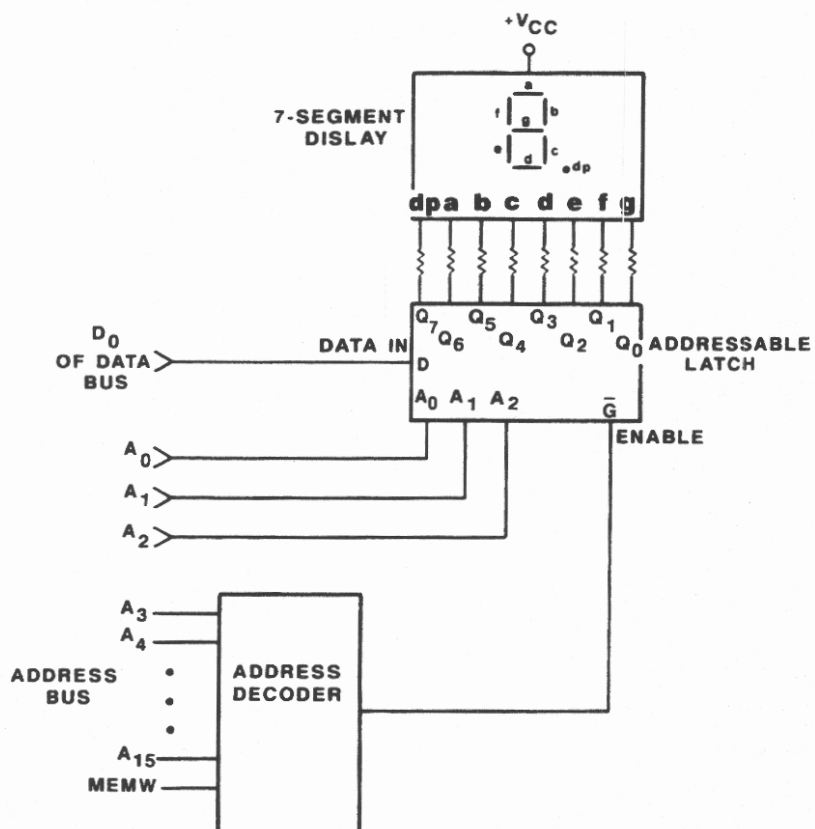


Figure 1-31

Using the addressable latch to drive a 7-segment display.

The only data line connected to the addressable latch is  $D_0$  from the microprocessor's data bus. The outputs of the eight latches ( $Q_0$  through  $Q_7$ ) drive the seven segments and the decimal point of the display.

While this arrangement results in an inexpensive circuit, it places an extra burden on the MPU. As in the previous example, the microprocessor must make the binary to 7-segment conversion. But in this case, it must also convert the parallel 7-segment code into a serial bit stream that is acceptable to the addressable latch. You are already familiar with the solution to the first problem. Now, the discussion will show how the MPU can convert parallel data to serial data.

Let's assume that the accumulator contains the 7-segment display code. Here's a subroutine algorithm that will output this code in serial form via the  $D_0$  data line to the addressable latch.

#### SUBROUTINE DISPLAY2

##### BEGIN

Load an index pointer register with the address of the latch (C160).

Load a counter register with the value 08.

##### NEXTBIT

Store the accumulator to the latch.

Rotate right the accumulator to get ready to send next bit.

Increment the index pointer register to the next address.

Decrement the counter register.

Jump if not zero to NEXTBIT.

Return

##### END.

The procedure is illustrated in Figure 1-32. The register on the left represents the accumulator. It contains the proper 7-segment code for displaying the letter 'A' via a common-anode display. This code must be transferred a bit at a time into the addressable latch shown on the right. Remember that the addressable latch actually contains eight latches and that the particular latch selected is determined by the 3-bit address on address lines  $A_0$  through  $A_2$ .



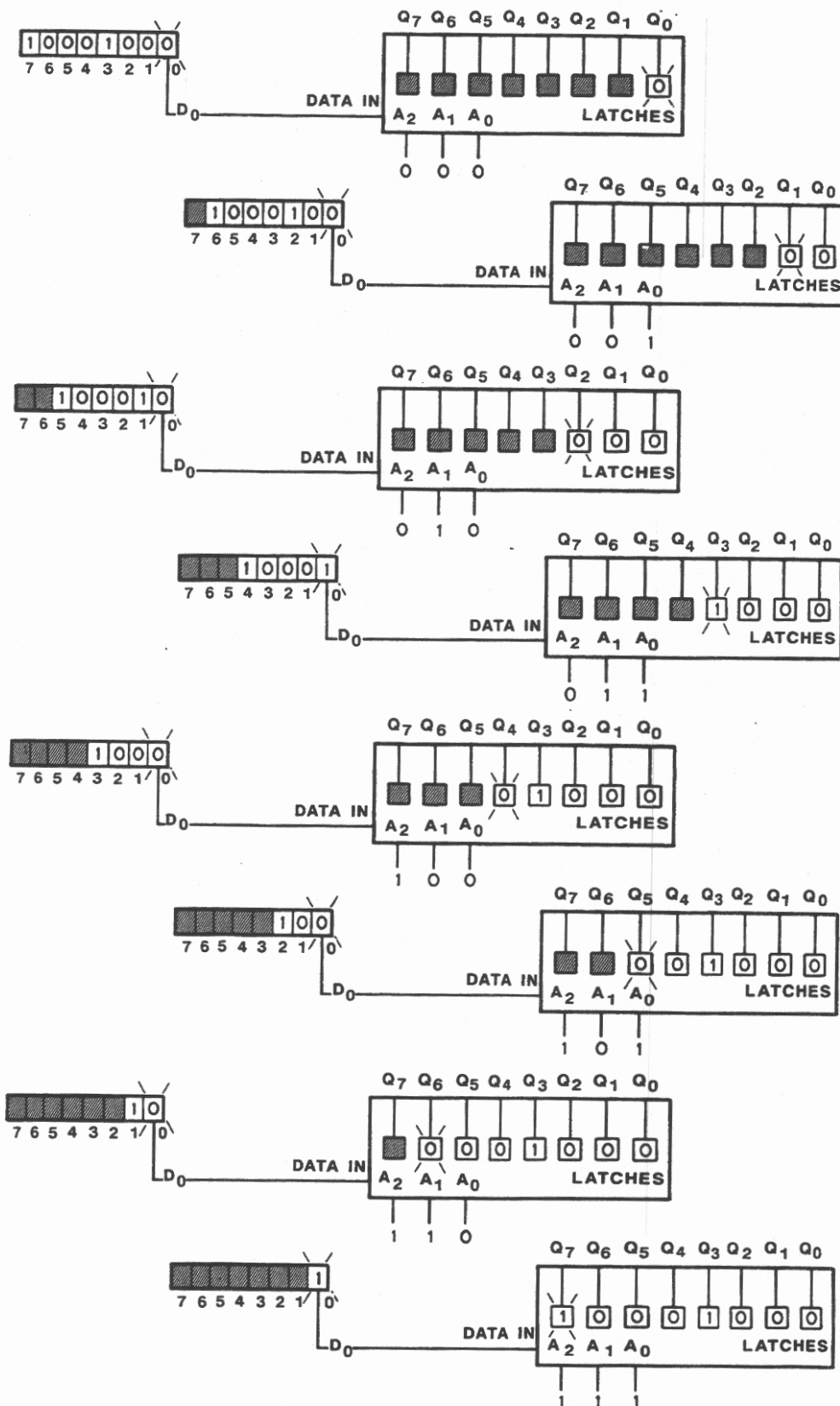


Figure 1-32  
Loading the addressable latch.

The algorithm first loads an index pointer register with the beginning latch address (C160), and initializes a counter register with the value 08. The first step is to store the accumulator to the basic latch address, C160. If you convert this address to binary, you will find that address bits  $A_0$ ,  $A_1$ ,  $A_2$  are all 0's. Since these lines connect to pins  $A_0$ ,  $A_1$ , and  $A_2$  on the addressable latch, latch 0 is selected as shown. Notice that the only data line used is  $D_0$ . Thus, the 0 in bit 0 of the accumulator is stored to latch 0.

Next, the contents of the accumulator are shifted to the right so that the next bit is placed in bit 0 of the accumulator. The address of the latch is incremented by 1, the counter is decremented by 1, and the accumulator is stored to the latch. Thus, the second bit is stored in latch 1.

This process continues as shown until the counter reaches 0, which means that all eight bits have been shifted from the accumulator to the addressable latch.

Of course, an 8085 program can be written to implement this algorithm. Step through the program below and verify that it works. The program employs the 8085 DE register pair as the index pointer register and the C register as the counter register.

```
                PUSH D
                PUSH B
                LXI D,C160H
                MVI C,07H
NEXTBIT        STAX D
                RRC
                INR E
                DCR C
                JNZ NEXTBIT
                POP B
                POP D
                RET
```

## Multiplexing Displays

The previous approaches required one or more latches for each display. There is a method by which we can drive up to eight displays using only two 8-bit latches. However, it requires some additional components and a lot of microprocessor time.

A typical circuit is shown in Figure 1-33. Eight common cathode displays are used in this example. The display cannot light unless its associated transistor is turned on. The transistors are controlled by the contents of the digit select latch. In turn, this latch is loaded by the MPU.  $Q_1$  is turned on by placing 1 in bit 7 of the digit latch;  $Q_2$  is turned on by placing 1 in bit 6; etc. Generally, only one transistor at a time is turned on. A common procedure is to store 10000000 in the digit select latch and rotate the contents so that the 1 appears at each bit in turn.

The individual segments of each display are turned on by the segment latch. For example, to turn on the decimal point in the right-most display, the segment latch must contain a 1 at bit 7. At the same time, the digit select latch must turn on  $Q_1$ .

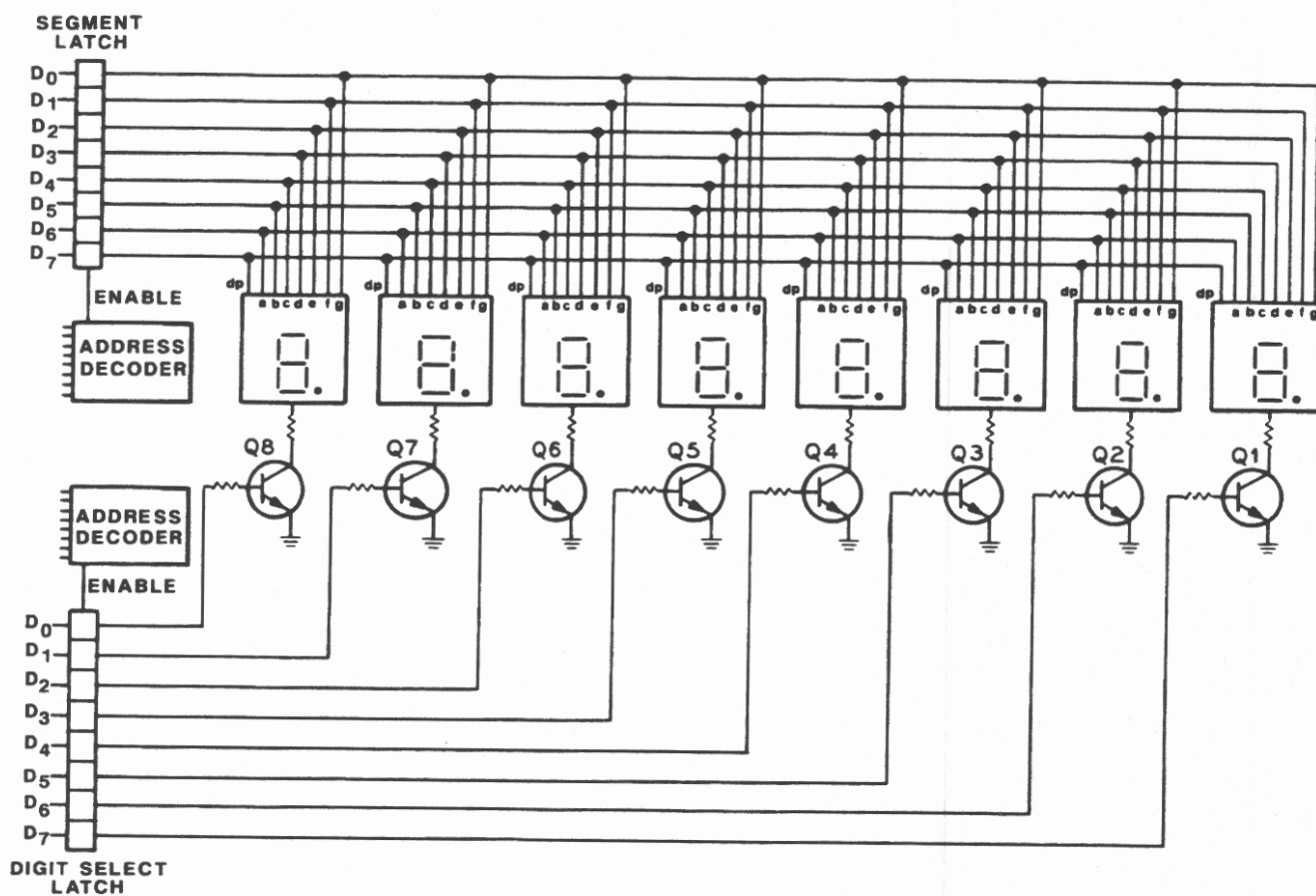


Figure 1-33  
Multiplexing Displays.

To display an 8-digit message, the procedure looks like this: Load the digit select latch with  $10000000_2$ . Load the segment latch with the 7-segment information for the right-most display. This causes the right-most display to show the first digit. After a brief delay, the contents of the digit display latch are changed to  $01000000$ . This turns on the second display. Next, the contents of the segment latch must be changed to the 7-segment code for the second display. This procedure continues until all eight displays are lit. Because no two displays are lit simultaneously, the displays must be refreshed many times each second to give the illusion of a constant steady display.

The advantage of this technique is that a minimum amount of hardware is required. However, it requires more MPU time than the previous techniques since the displays must be constantly refreshed.

## Self-Test Review

28. How do you turn on a common-anode type 7-segment display?
29. What is a disadvantage of using a decoder-driver, such as the 7447, to drive a display?
30. Why is a latch required between the MPU and the display?
31. When the latch drives the display directly, what performs the decoding function?
32. List four instructions that could be used to output data to a display.
33. Using the arrangement shown in Figure 1-27, what 8-bit code is required to form the letter P?
34. Refer to Figure 1-28 and the DISPLAY1 subroutine. If the number in the accumulator is  $08_{16}$  when this program segment is run, what binary number will be outputted to the display?
35. What is the purpose of the DISPLAY1 subroutine?
36. Refer to Figure 1-31. What determines which of the latches the input data bit goes into?
37. Subroutine DISPLAY2 converts \_\_\_\_\_ data to \_\_\_\_\_ data.
38. Refer to Figure 1-33. What determines which character is displayed?
39. Refer to Figure 1-33. What determines the display on which the character appears?

## Answers

28. A common-anode type display is turned on by applying logic 0 to the proper cathode.
29. The pattern is limited to those provided by the decoder.
30. Because the output data is stable for only an instant, a latch must be used to capture the data and hold it for the display.
31. The microprocessor.
32. Many of the move or store instructions could be used to output data to a display. These include: MOV r, M, MVI M, STA, STAX.
33.  $0110\ 0111_2$ .
34.  $0111\ 1111_2$ . This is the 7-segment pattern for the numeral 8.
35. The DISPLAY1 subroutine converts binary numbers between 0000 and 1111 to a 7-segment code to form the appropriate hexadecimal numeral.
36. The 3-bit address  $A_0$  through  $A_2$ .
37. parallel, serial.
38. The value in the segment latch.
39. The value in the digit select latch.

## UNIT SUMMARY

1. In computer jargon, a bus is generally defined as a group of conductors that transfer information in parallel from. The data bus on the 8085 MPU is eight lines wide. Thus, it can transfer eight bits of data at one time.
2. Information transfers on a bus can be bidirectional or unidirectional. The data bus is bidirectional, while the address bus is unidirectional.
3. Many devices can share a bus network.
4. An address decoder in a microcomputer can be as simple as a couple of NAND and NOR gates wired together or as complex as a dedicated address decoding device.
5. Three-state logic gates, or buffers, are used on a bus to control which devices can communicate on a bus at any point in time. These devices can be bidirectional or unidirectional.
6. What? Where? How? and When? The MPU data bus signals indicate *what* information is being transferred, the address signals indicate *where* the data is being transferred to or from, and the control bus signals indicate *when* and *how* the data is to be transferred.
7. I/O lines  $AD_0$  through  $AD_7$  provide a multiplexed low order address and data bus. These lines must be demultiplexed using an external latch and the  $ALE$  control signal (See Figure 1-13).
8. The  $S_0$ ,  $S_1$ , and  $IO/\overline{M}$  lines are output status lines that are used to identify various operations to external peripheral devices.
9. Serial I/O can be achieved from the 8085 accumulator via the  $SID$  and  $SOD$  lines.
10.  $\overline{RESET}$  is the highest priority interrupt. When activated,  $\overline{RESET IN}$  causes the MPU to clear the program counter, and tri-state its bus lines. At the same time, the  $RESET OUT$  line goes high to reset external devices.
11.  $INTR$  (Interrupt Request) is a maskable interrupt that requires external logic to generate a hardwired RST (restart) instruction to locate the interrupt service routine. There are eight RST instruction possibilities that cause the MPU to go to eight unique call locations where a short service routine or JUMP instruction must be located.

12.  $\overline{INTA}$  (Interrupt Acknowledge) is the response to an  $INTR$  input. The  $\overline{INTA}$  signal is used to enable an external hardwired 3-state buffer to generate a  $RST$  instruction onto the data bus.
13.  $TRAP$  is a non-maskable interrupt that is activated by a rising edge/level. A  $TRAP$  interrupt causes the program counter to be stacked (pushed) and the MPU to vector to address  $0024_{16}$  where a short service routine or  $JUMP$  instruction must be located. A  $RET$  instruction at the end of the service routine causes the program counter to be unstacked (popped) and the MPU to return to where it left-off before the interrupt.
14. The  $RST\ 5.5$ ,  $6.5$ , and  $7.5$  interrupts are all maskable interrupts. They are enabled and disabled using a  $SIM$  instruction. The  $SIM$  instruction reads the accumulator and enables/disables the interrupts according to the current bit status of the accumulator (See Figure 1-16). The  $RST\ 7.5$  interrupt is edge triggered, while the  $RST\ 5.5$  and  $RST\ 6.5$  interrupts are level triggered.
15. The  $\overline{IOR}$ ,  $\overline{IOW}$ ,  $\overline{MEMR}$ , and  $\overline{MEMW}$  I/O control signals are obtained from the 8085  $IO/\overline{M}$ ,  $RD$ , and  $WR$  lines using external logic (See Figure 1-18).
16. Direct I/O requires that the lower address bus be decoded along with either the  $\overline{IOR}$  or  $\overline{IOW}$  control signal to create an input or output port. The  $IN$  and  $OUT$  instructions are employed to input and output data, respectively, between the accumulator and I/O device ports.
17. Memory-mapped I/O requires that the entire address bus be completely or partially decoded along with either the  $\overline{MEMR}$  or  $\overline{MEMW}$  control signal to map the I/O devices within the 8085 memory map space. Any of the 8085 memory-reference instructions can be used to communicate via any of the MPU registers and the memory-mapped I/O devices.
18. The most popular input device used with the microprocessor is the switch. The more common switches are found in keyboards, however, many forms of automated equipment use limit switches, pressure switches, etc.
19. Interfacing with a switch involves four MPU operations. First, the MPU must address the proper switch. Second, it detects contact closure. Third, it usually has to debounce the switch signal (may be a hardware operation). Fourth, it must decode the switch signal.



20. One simple method for interfacing individual switches to the MPU is to assign a buffer to a memory location, and then connect each switch to a buffer input. Up to eight switches can be connected in this manner. To read the status of an individual switch: the MPU addresses the buffer, transfers its contents to an accumulator, and masks all but the desired bit. The MPU may have to perform this operation several times and compare the results to eliminate the possibility that contact bounce is affecting the switch status.
21. When many keys must be interfaced, such as in a keyboard, they are usually arranged in a matrix. For example, a 64-key keyboard would be arranged in eight columns and eight rows. The eight columns are assigned individual addresses and the eight rows are coupled through a buffer to a single address. To determine which key has been pressed, each column is sequentially activated and the output from the eight keys in the selected column are coupled to the buffer and read into the accumulator. Assuming the pressed key outputs a logic zero, the eight bits are tested for a logic zero. Each column is activated and tested until a logic zero is found. The MPU uses a "counter" to determine the location of the closed key.
22. There are many methods for driving LEDs. For decimal display, a BCD-to-7-segment decoder-driver translates the 4-bit BCD input to the proper patterns to drive the seven LEDs segments. For hexadecimal display, the best scheme is to drive the LED directly from the MPU through an 8-bit bistable latch. The latch can be treated as a single memory location. An alternate method is to use a serial input, 8-bit addressable latch to drive the LEDs.
23. To display messages, you must multiplex several 7-segment LEDs. The MPU software turns-on one display at a time, while storing the display value to the display. The displays are rapidly scanned in this manner to prevent flicker.
24. Up to eight LED displays can be driven by two 8-bit latches and eight driver transistors, if the MPU multiplexes the display data. Compared to the other methods, this system uses a minimum amount of hardware, but it does require more software and MPU time.

*Unit 2*

**MEMORY**

## CONTENTS

Introduction .....	2-3
Unit Objectives .....	2-4
Interfacing to RAM .....	2-5
Static RAM Interfacing .....	2-6
Connecting Static Ram to the MPU .....	2-9
Dynamic RAM Interfacing .....	2-12
Dynamic RAM Refresh .....	2-18
Dynamic RAM Control .....	2-19
Connecting Dynamic RAM to the MPU .....	2-23
64K and Larger RAMs .....	2-23
Self-Test Review .....	2-25
Answers .....	2-27
EPROM Programming and Interfacing .....	2-29
A Typical EPROM Architecture .....	2-30
Programming the EPROM .....	2-34
Self-Test Review .....	2-35
Answers .....	2-36
The ETW-3800 (8085) Trainer Memory Map .....	2-37
I/O Memory Map .....	2-39
Unit Summary .....	2-40

## ***Unit 2***

# **MEMORY**

## **INTRODUCTION**

As you are now aware, the complete microcomputer system requires various forms of memory to be completely functional. Read/write memory, or RAM, is used to temporarily store programs and data; read-only memory, or ROM, is used for nonvolatile storage of frequently used routines; and mass secondary memory devices, such as magnetic tapes and disks, are used for long term non-volatile storage.

In this unit, you will learn how to interface various forms of RAM and ROM to the MPU. You will learn the fundamentals required to interface static RAM, dynamic RAM, and EPROM. In addition, you will learn to interface a static RAM and program an EPROM. Finally you will learn about the internal memory features of the 8085 microprocessor trainer through an examination of its memory map.

## UNIT OBJECTIVES

1. Explain the internal structure and operation of a static RAM device.
2. Describe how to interface to static RAM devices.
3. Explain the internal structure and operation of a dynamic RAM device.
4. Describe how to interface to dynamic RAM devices.
5. Describe the software and hardware requirements of dynamic RAM refresh and control.
6. Describe the hardware and software required to program an EPROM.
7. Explain how to interface to a ROM device and read its contents.
8. Describe the memory map of the 8085 microprocessor trainer.

## INTERFACING TO RAM

Random access memory is any memory that lets you directly access any storage cell location. Thus, you could apply this term to read-only memory or read/write memory. However by convention, read/write memory is considered to be RAM.

There are two basic types of RAM, dynamic and static. Dynamic RAM stores each bit of data as a charge level in a MOSFET cell that acts like a capacitor. Because of capacitor leakage, the charge must be replenished periodically. This is known as *data refresh*. Therefore, in addition to the RAM device itself, you need additional circuitry to handle the refresh. Static RAM, on the other hand, uses bistable flip-flops to store data. Since data is latched in these flip-flops, no refresh is required. That is, data can be maintained indefinitely without refreshing, as long as power is applied to the device.

In this section, we will review the principles of both static and dynamic RAM. Then we will examine a specific interfacing example of each.

## Static RAM Interfacing

Static RAM, as mentioned earlier, uses an integrated flip-flop for each of its storage cells. These flip-flops can be constructed from either bipolar transistor or MOSFET technology. The actual method used will depend on the system requirements. The primary factors involved are speed, storage density, and power consumption. However, because of a continuous improvement in design and the unique characteristics of specific devices, an in-depth study of RAM selection is beyond the scope of this discussion.

While the technologies may vary, the application is essentially the same. Therefore, we'll look at the more popular MOSFET-type RAM. Figure 2-1 shows a typical RAM storage cell.

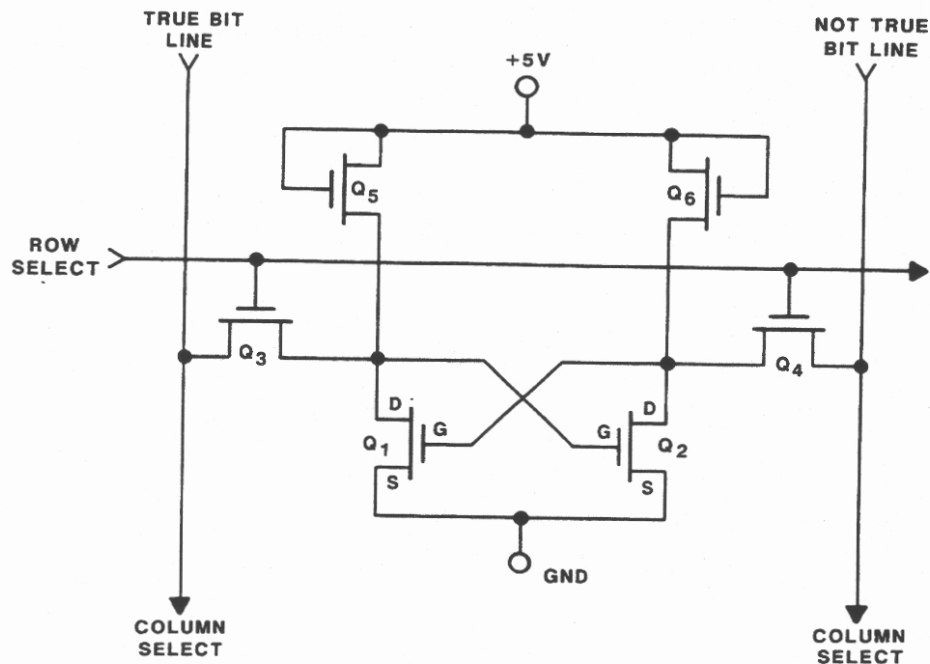


Figure 2-1  
Static RAM MOSFET storage cell.

The storage cell consists of six MOSFETs, Q1 through Q6. Q1 and Q2 are cross-coupled to form the bistable flip-flop or latch that stores the bit of data. Q5 and Q6 act as "load resistors" for Q1 and Q2 respectively. Finally, data is coupled to and from the latch through Q3 and Q4. A logic 1 on the row select line turns Q3 and Q4 of each cell in that row on. Then it's up to the column select lines to determine which cells are accessed.

Notice that the RAM cell in Figure 2-1 uses two column select lines to couple bit-size data to and from the flip-flop. Because of the flip-flop's configuration, both the data bit and its complement must be latched. For that reason, the column select lines are also called the "true" and "not true" bit lines. Since only *true* logic data is supplied to the RAM, an internal circuit called "Column I/O" generates the necessary *not true* data. This and the other RAM control circuits are shown in the block diagram in Figure 2-2. The figure illustrates a typical RAM IC.

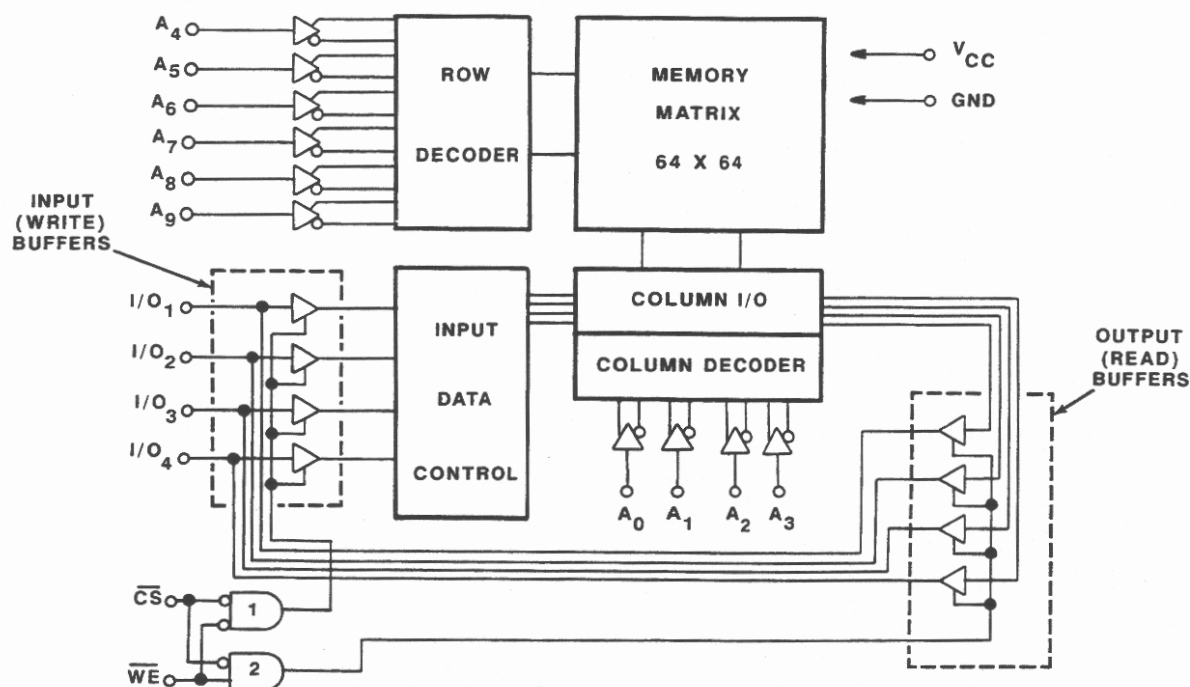


Figure 2-2  
Block diagram of a typical RAM IC.



For ease of explanation, we have chosen to use a  $1K \times 4$ -bit RAM. Internally, it is made up of an array of 64 storage cell rows and 64 storage cell columns. High-order address lines A4 through A9 access the rows, while low-order address lines A0 through A3 access four specific cell columns in that row. The 64 columns are subdivided into groups of 16, in this case there are four groups, as shown in Figure 2-3. The four low-order address lines then select a single column in each subgroup. In this manner, four specific storage cells are accessed for a 4-bit memory read or write.

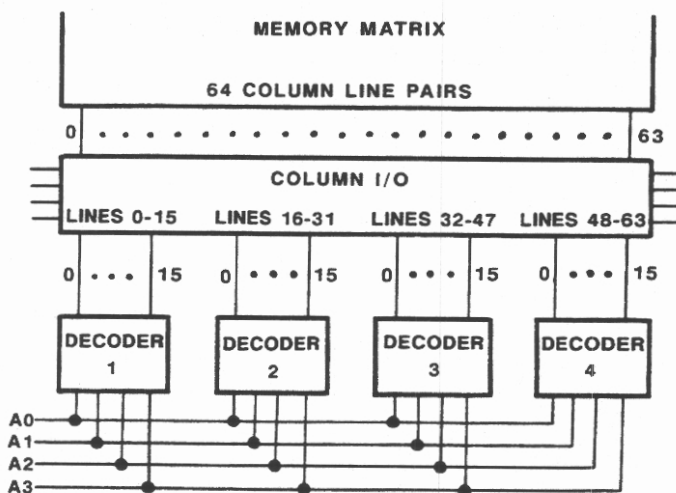


Figure 2-3

Dividing the 64 column select lines into groups of 16.

Notice in Figure 2-2 that each address line is connected to a buffer with two outputs. This is illustrated simply to show that the address value and its complement are used by the Row and Column Decoder circuits. They have nothing to do with the actual row and column select signals.

Bidirectional, 3-state buffers isolate the RAM from the data bus. Two cross-coupled gates use the Chip Select and Write Enable control signals to determine which set of four data buffers is enabled. If  $\overline{CS}$  is high, the output of each gate will be forced low, disabling both the input and the output buffers. However with  $\overline{CS}$  low, the Write Enable line determines data direction. With  $\overline{WE}$  low, gate 1 will enable the input (write) buffers, while with  $\overline{WE}$  high, gate 2 will enable the output (read) buffers.

The last two areas not covered thus far are the Column I/O and Input Data Control circuits. Basically, they handle the reading and writing of data inside the RAM. Sense amplifiers in the Column I/O circuit couple data from the selected memory cells to the output buffers. Input data is first amplified in the Input Data Control circuit and then coupled to the four selected memory cells by the Column I/O.

## Connecting Static Ram to the MPU

Although a wide variety of static RAMs are available today, for discussion we'll use a  $1\text{K} \times 8$  bit arrangement of RAM chips. The principles of interfacing other size RAMs are similar. Figure 2-4 is a partial schematic of a typical microprocessor system. Two RAM IC's are used as a  $1024_{10}$ -by-8 RAM. Even though not shown, the address and data buses from the MPU are also connected to a ROM, input and output circuits, and possibly additional RAMs. The MPU can communicate with only one of these devices at any one time. To communicate with the two IC's, the MPU must first select them by switching their  $\overline{\text{CE}}$  lines low. Notice that the  $\overline{\text{CE}}$  lines are connected to the output of the RAM address decoder. This decoder monitors the six high-order address lines. When the address of the RAM appears on these lines, the two RAM chips are enabled.

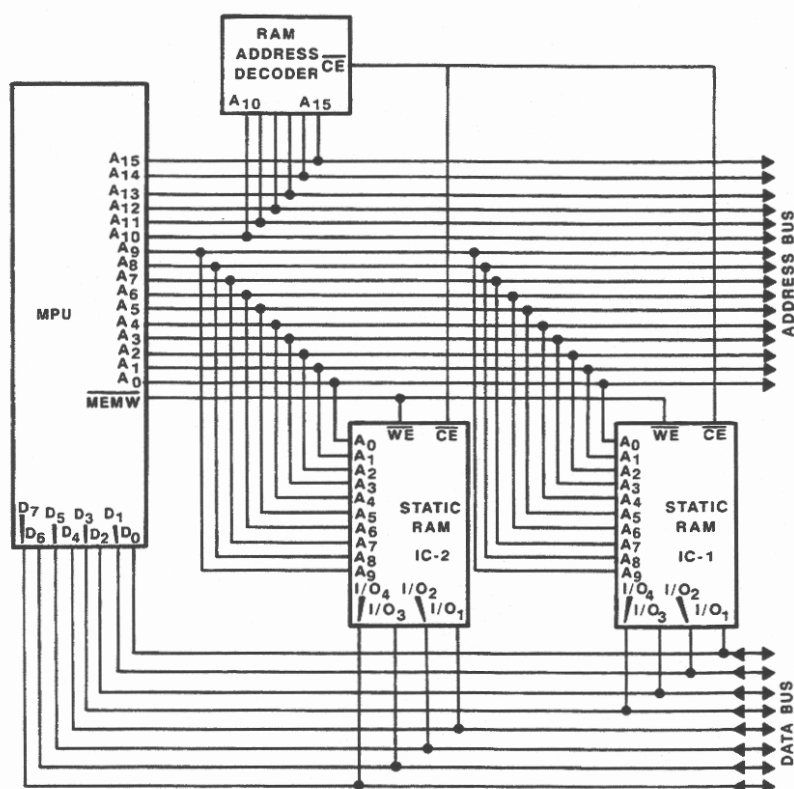


Figure 2-4

Partial schematic of a typical microprocessor using two static RAMs.

The two RAM IC's make up a  $1024_{10}$  by 8-bit RAM. IC1 connects to the four least significant bits of the data bus ( $D_0$  through  $D_3$ ), while IC2 connects to the four most significant bits. Thus, when a byte of data is stored in the RAM, the four LSB's go in IC1, while the four MSB's go in IC2. This is possible because the two IC's are enabled at the same time. Notice that the address,  $\overline{\text{CE}}$ , and  $\overline{\text{WE}}$  lines of the two ICs are tied together.

The address lines of the IC's monitor the  $A_0$  through  $A_9$  lines from the MPU. Once the chips are enabled, any one of the  $1024_{10}$  memory locations can be selected by placing the proper address on the lower ten address lines.

The  $1024_{10}$ -byte RAM must be assigned some starting address. In a typical system, a common practice is to assign RAM the lowest addresses, although it may be assigned elsewhere. Thus, a  $1024_{10}$ -byte RAM would be given address  $0000_{16}$  through  $03FF_{16}$ . In binary, these are addresses  $0000\ 0000\ 0000\ 0000_2$  through  $0000\ 0011\ 1111\ 1111_2$ . Notice that the ten LSB's can specify any one of the  $1024_{10}$  memory locations. However, it is the upper six bits that specify that the starting address is at the low end of memory. That is, the RAM must be enabled when the upper six bits of the address bus are  $0000\ 00$ .

In Figure 2-4, the RAM address decoder monitors the upper six bits of the address bus. When this decoder finds that the upper six bits are all zeros, it switches the  $\overline{CE}$  line low, enabling the RAM.

The address decoder can be any type of logic circuit that meets the above requirements. A typical circuit is shown in Figure 2-5. If you trace through the various logic levels, you will see that  $\overline{CE}$  is low only when  $A_{10}$  through  $A_{15}$  are low. The inputs to NOR gate 1 and the inputs to NOR gates 2 are low when the high-order address is  $0000\ 00_2$ . The two NOR gates produce high outputs. Thus, the inputs to NAND gate 3 are all high. This forces the output of gate 3 low. As you can see, this circuit fulfills the requirements of the address decoder.

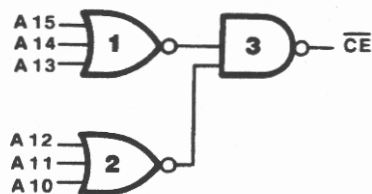


Figure 2-5  
Address decoder using discrete logic gates.

A memory location in the RAM shown in Figure 2-4 is selected by all 16 address lines. The low-order address lines connect directly to the RAM IC's, while the high-order lines connect to the RAM address decoder. Thus, each memory location in RAM has only one address. The addresses are said to be *fully decoded*.

We can save some decoding logic by only *partially decoding* the address. Figure 2-6 shows an address decoder that monitors only two of the address lines ( $A_{14}$  and  $A_{15}$ ). If you trace the logic levels through, you will see that  $\overline{CE}$  is low any time that  $A_{14}$  and  $A_{15}$  are low.  $A_{14}$  and  $A_{15}$  will be low for any address at or below  $3FFF_{16}$ . If the  $\overline{CE}$  line is used to enable a RAM that has fewer than  $3FFF_{16}$  bytes, some of the addresses will be duplicated. To illustrate this point, assume that we replace the address decoder shown in Figure 2-4 with the circuit shown in Figure 2-6. Memory location  $0000_{16}$  can be selected by placing address  $0000_{16}$  on the address bus. However, location  $0000_{16}$  is also selected when  $0800_{16}$  is placed on the address bus. The reason for this is that the RAM is enabled because  $A_{14}$  and  $A_{15}$  are low. And, the lowest address in the RAM is read out because  $A_0$  through  $A_9$  are low. Actually, there are dozens of addresses that will select any given location. For example, addresses  $3C00_{16}$ ,  $2400_{16}$ ,  $1800_{16}$ , and many others will all select memory location  $0000_{16}$ . This is the price that must be paid for saving a few gates in the address decoder.

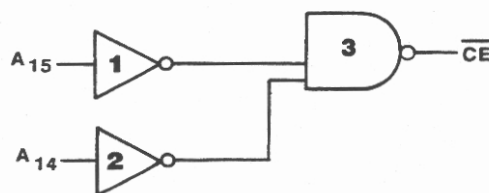


Figure 2-6

We can save gates by only partially decoding the address.

In practice, this does not actually cause many problems. All we have done is sacrifice the lower  $16,384_{10}$  addresses to  $1024_{10}$  bytes of RAM. However, this still leaves three-fourths of the  $64K$  of addresses untouched. In most cases, this leaves more than enough addresses for I/O devices, ROMs, etc.

The fact that a given byte of data appears at several addresses is also no problem as long as we remember this limitation in our programming. Partial decoding schemes are frequently used because they save decoding logic.

You should be aware that static RAM devices are available that can store upwards of  $64K$  bytes of data in a single IC. You will work with an  $8K \times 8$  RAM in an associated Experiment.

## Dynamic RAM Interfacing

Memory systems built around static RAM appear to be the ideal approach for a microcomputer. Static RAM is fast, there are many configurations to choose from, and best of all, it needs no refresh circuitry. So why bother with dynamic RAM?

There are, in fact, two very good reasons. First, you can package more storage cells into a dynamic RAM IC than you can in a static RAM IC. Figure 2-7 shows you why. A dynamic RAM storage cell consists of one MOSFET and a capacitor. The MOSFET and capacitor occupy about the same space as two of the static cell MOSFETs. Thus, three dynamic storage cells can fit into about the same space as one static storage cell. This means that you will essentially cut the amount of circuit board space used for memory by two-thirds, a substantial savings.

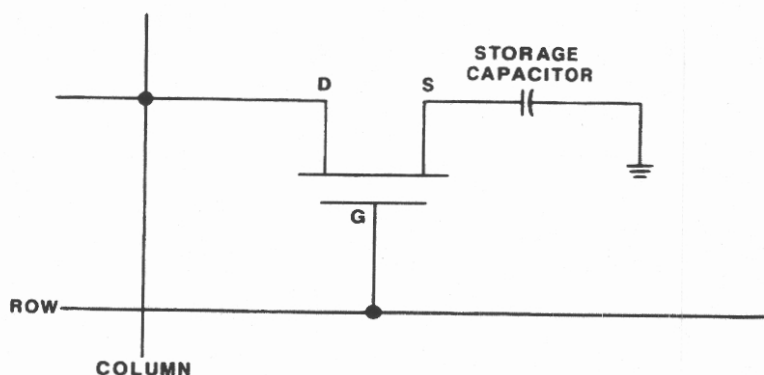


Figure 2-7  
Single dynamic RAM storage cell.

The second reason for using dynamic RAM is power dissipation. Power dissipation is the amount of power, in watts per storage cell, required to operate the memory device. Dynamic RAM has lower power dissipation because there are fewer active devices (MOSFETs) per cell.

Naturally, we can't forget the additional circuitry needed to handle memory refresh. This can offset the savings from using dynamic RAM, in terms of board space and power. So there are certain trade-offs that must be considered before you select one type of RAM over the other. Chief among these are available system space, power supply limitations, present memory requirements, and further memory expansion. However, as a general rule of thumb, use static RAM if your system requires less than 16K of memory, and dynamic RAM if your system needs more memory.

Functionally, the dynamic RAM storage cell shown in Figure 2-7 employs the charging/discharging action of a capacitor. The capacitor is charged or discharged to indicate bit logic. If the capacitor is charged, a logic 1 is stored in the cell. A discharged condition indicates a logic 0.

Writing data to a dynamic RAM storage cell involves two operations. First, the row address is decoded and the specified row accessed. This causes every cell on that row to turn on. Next, data to be written to each cell is placed on the appropriate column lines. If the data is a logic 1, the storage cell capacitor is charged; a logic 0 leaves the capacitor discharged.

Reading data from a dynamic RAM storage cell involves the same basic process. First, the appropriate row is accessed, turning on every MOSFET in that row. As soon as each cell MOSFET is turned on, the current stored in its capacitor, if the capacitor is charged, is coupled to the cell column line. This current flow, or lack of current flow, is sensed by the column sense amplifier as a logic 1 or a logic 0 respectively. Notice, however, that a read operation discharges the capacitor of every cell on the accessed row. This is common to all dynamic RAMs, and is called a **destructive read**. To restore each cell to its original logic level, a special circuit in the column sense amplifier monitors the logic level and then writes that value back to the cell. Later you'll see how this read and restore process is used for memory refresh.

Dynamic RAM is available in different size memory packages ranging from 1K to over 256K bits in density. Except for a few minor exceptions all dynamic RAM is bit-wide memory. That is, it would take eight RAM packages to store a byte of data. Figure 2-8 shows the block diagram and pin configuration of a typical dynamic RAM device. This is a 16K x 1 bit memory circuit.

Shown in the figure is a 16-pin package that is a standard for most 16, 32, and 64K dynamic RAM. Three of the pins connect to +5V, -5V, and +12V power, with a fourth connected to power supply ground. (Note that -5V and +12V are not always used in this RAM package.) Seven other pins are address lines. The  $\overline{RAS}$  and  $\overline{CAS}$  pins are part of the address decoding network and will be discussed later. The  $D_{IN}$  and  $D_{OUT}$  pins are separate data input and 3-state output pins. In most cases, these two pins are tied together, and then connected to the data bus. Later, we'll look at an example where these pins aren't tied together. The last pin,  $\overline{WE}$  (Write Enable), determines data direction. A logic high on this pin indicates a *read* operation, while a logic low indicates a *write* operation.

The internal structure of the dynamic RAM maps the 16K 1-bit storage cells into an array consisting of 128 rows and 128 columns. As the block diagram in Figure 2-8 shows, the rows are divided into groups of 64, with the 128 column sense amplifiers positioned between the two row groups. Seven address bits are needed to address any row. This data is stored in the "7-Bit Row Latch." A second 7-bit latch stores the seven address bits required to address a column. Because of certain design constraints, there are only 64 column decoders for the 128 sense amplifiers. Thus, the final decoding is handled by the I/O gating circuit. For that reason, only six of the column address bits are used by the column decoders. The seventh bit ( $A_6$ ) is used by the I/O gating circuit for final decoding.

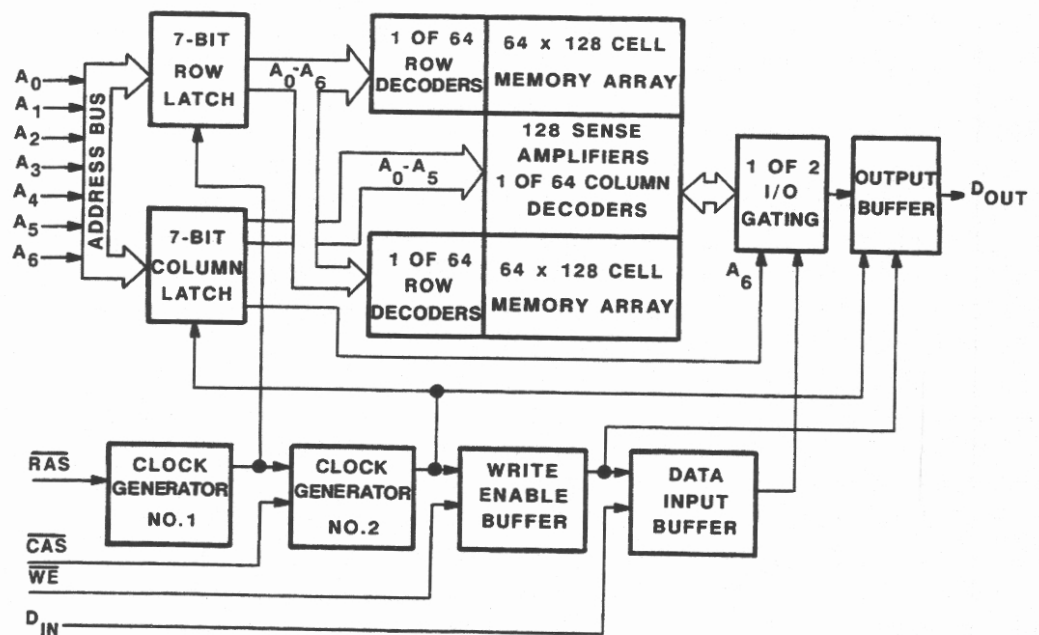


Figure 2-8

Block diagram and pin configuration of a 16K x 1 dynamic RAM.

Notice that it takes a total of 14 address bits to access one storage cell within the dynamic RAM. Yet it has only seven address input lines, A0 through A6. The problem is solved by multiplexing the address information. First the row address is latched, then the column address is latched. This requires an external *address multiplexer* and two control signals, *row address strobe (RAS)* and *column address strobe (CAS)*, as shown in Figure 2-9. Figure 2-10 shows the timing for storing an address to memory.

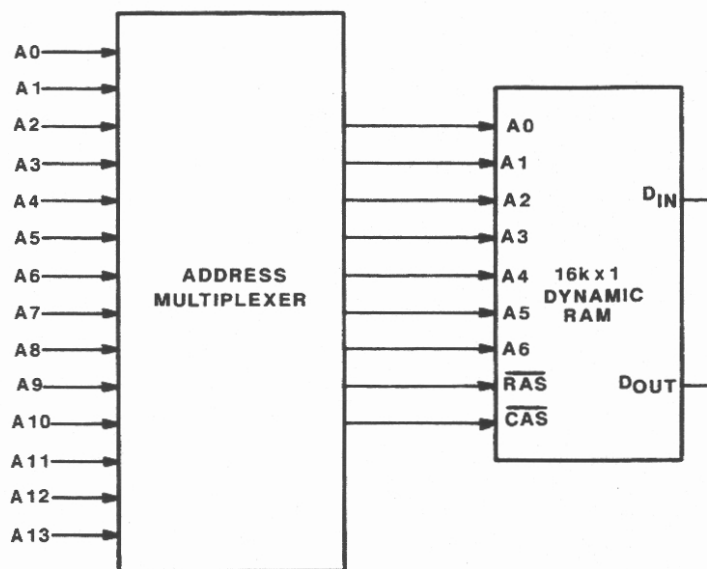


Figure 2-9  
Address multiplexing for the 16K x 1 dynamic RAM.

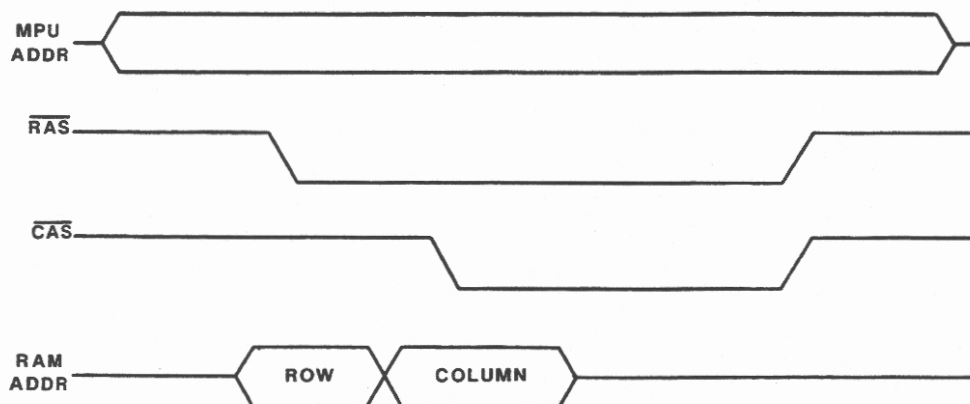


Figure 2-10  
Multiplexed RAM address timing.



As soon as the multiplexer receives a valid 14-bit address, it places the seven low-order bits,  $A_0$  through  $A_6$ , on RAM address lines  $A_0$  through  $A_6$ . A few nanoseconds later, it pulls  $\overline{RAS}$  low. As  $\overline{RAS}$  goes low, the "7-Bit Row Latch" inside the RAM is enabled, and the row address is latched. A short period after  $\overline{RAS}$  goes low, the multiplexer replaces the low-order address bits with high-order address bits  $A_7$  through  $A_{13}$ . A few nanoseconds later,  $\overline{CAS}$  is pulsed low. This causes the high-order address bits to be latched into the "7-Bit Column Latch." With the RAM row and column addresses latched, the decoding circuits can access the specified storage cell.

Referring back to Figure 2-8, notice how the row and column address strobes are routed inside the IC. Each toggles a "clock generator," which in turn toggles the appropriate latch. However, as the signal flow arrows indicate, Clock Generator No. 2 cannot respond until after Clock Generator No. 1 is toggled. This is a safety feature to make sure the RAM responds to the  $\overline{RAS}$  signal before it responds to the  $\overline{CAS}$  signal.

In addition to controlling the 7-Bit Column Latch, Clock Generator No. 2 also sends an enable signal to the Output Buffer. Thus, if the system is performing a read operation, storage cell data will be coupled through the buffer to the  $D_{OUT}$  pin. Naturally, during a read operation, both the Write Enable Buffer and Data Input Buffer will be disabled.

For a write operation, the high-to-low transition on the  $\overline{WE}$  line toggles the Write Enable Buffer (Figure 2-8). This in turn disables the Output Buffer, and enables the Input Buffer. With the Input Buffer enabled, data at the  $D_{IN}$  pin is coupled through the I/O Gating and Column Decoder circuits to the selected storage cell.

A read operation can occur in only one fashion; the row address is latched, then the column address is latched and the output buffer is enabled. A write operation, on the other hand, is not fixed to one sequence of events. It can go through what we call a **normal**, or early, write cycle or it can go through a delayed, or late write, cycle. Both are tied to the timing of  $\overline{CAS}$  and  $\overline{WE}$ . Figure 2-11 shows the differences.

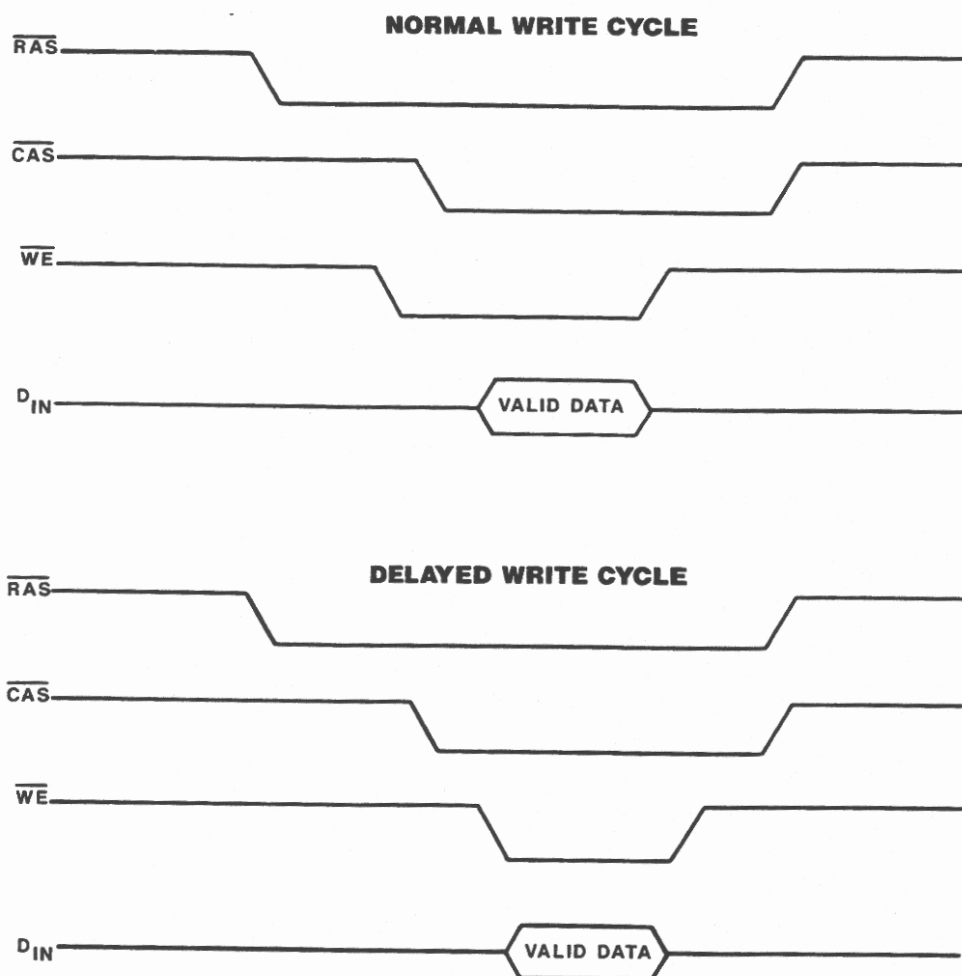


Figure 2-11

Signal timing for both the Normal and the Delayed Write Cycle.

In the normal write cycle,  $\overline{WE}$  goes low before  $\overline{CAS}$  goes low. Thus, the input data is strobed by the negative transition of  $\overline{CAS}$ . Also, the RAM setup and hold times are referenced to  $\overline{CAS}$ . In addition, bringing  $\overline{WE}$  low first guarantees that the output buffer is disabled ( $D_{OUT}$  floating).

The opposite action takes place in a delayed write cycle;  $\overline{CAS}$  goes low before  $\overline{WE}$ . Now the input data is strobed by the negative transition of  $\overline{WE}$ . Changing the signal sequence will have no effect on the actual "write" process, other than referencing the setup and hold times to  $\overline{WE}$ . However, if  $\overline{CAS}$  goes low before  $\overline{WE}$ , the output buffer is not disabled. This means that you cannot connect the  $D_{IN}$  pin directly to the  $D_{OUT}$  pin in the delayed write mode of operation.

An extension of the delayed write operation in a dynamic RAM involves what we call the **read-write**, or **read-modify-write**, cycle. It is a simple method of verifying and correcting data stored in memory. It works in this manner: First, a read operation is performed on a bit in memory. If the data is correct, the cycle is ended. However, if the data is incorrect, the  $\overline{WE}$  pin is pulled low and the correct data is written to the cell. All of this takes place while  $\overline{RAS}$  and  $\overline{CAS}$  are held low. Thus, you only have to address the storage cell once to perform both a read and a write operation.

## DYNAMIC RAM REFRESH

Recall that dynamic RAM must be refreshed periodically. With many dynamic RAM ICs, refreshing requires that each row of the RAM be addressed at least once every two milliseconds. The columns do not have to be addressed, and you do not have to perform a read-write cycle on each bit within the RAM. To refresh a bit, you simply address its row. Addressing one row in a 16K RAM will refresh all 128 bits in that row. Recall from an earlier discussion that when a row is addressed, the data is destructively read from each cell in that row. Then, a special circuit in each column sense amplifier writes the same data back to each cell. This is how the cells in a row are refreshed. Thus, addressing all 128 rows every two milliseconds will refresh all 16K bits of a 16K dynamic RAM. Reading data from, or writing data to a cell will cause all of the cell in the addressed row to be refreshed. However, unless you know that you will be repeatedly accessing all of the rows during the normal read/write process, you must plan for a specific refresh operation.

There are two general methods of refreshing dynamic RAM. They are **distributed**, or **asynchronous**, refresh and **hidden**, or **synchronous**, refresh. Each has advantages over the other; your choice will depend on your system requirements. This includes the type of memory controller used. The memory controller can be a single multifunction device or a specialized circuit containing discrete gates, multiplexers, and timers. In either case, the controller handles address multiplexing,  $\overline{RAS}$ ,  $\overline{CAS}$ , and  $\overline{WE}$  timing, and memory refresh. We'll discuss the memory controller later; but for now, let's look at the two types of refresh.

In a "distributed refresh" system, the memory controller periodically generates an internal request for a refresh cycle, typically every 10-16 microseconds. Since the refresh request is asynchronous to the MPU's request for memory, the memory controller must have logic to arbitrate the two requests. Once a bus cycle or a refresh cycle starts, the arbiter must let that cycle finish before starting a pending cycle. This means that the memory controller must also have circuitry that can force the MPU to "wait" if a bus cycle is requested during a refresh cycle.

"Hidden refresh" designs use circuitry to monitor MPU timing and request a refresh cycle when the MPU is not executing a RAM bus cycle. For example, a hidden refresh cycle can overlap an instruction fetch from ROM. The reason why the refresh cycle won't interfere with a ROM bus cycle is that the RAM address lines are isolated from the MPU address lines by the RAM controller. As long as the hidden refresh cycles are performed frequently enough, the dynamic memory is always ready when the MPU requests a RAM bus cycle.

A third type of memory system uses a combination of distributed and hidden refresh. For example, some types of systems allow an MPU to enter a HALT state (which stops program execution) while waiting for an interrupt. During this period, the hidden refresh circuitry is also inactive. Should this happen, the distributed refresh circuitry "times-out," and generates its own refresh cycle. This "automatic" refresh will continue until the MPU exits its HALT state and resumes normal operation. At that point, the hidden refresh circuit will again take over.

## DYNAMIC RAM CONTROL

Dynamic RAM control can be handled by a circuit composed of many discrete components. This route is often chosen because it gives the design engineers more control over the memory system and its future expandability. However, due to the complexity of this type of RAM controller, a detailed description is not practical. On the other hand, the single-chip 8203 Dynamic Memory Controller accomplishes the same basic tasks, and will serve as a good example for interfacing with dynamic RAM. The functions we will be describing for the 8203 Dynamic RAM Controller deal primarily with its operation as a 16K dynamic RAM controller.

In addition to these functions, the 8203 can control four banks of 4K dynamic RAM or two banks of 64K dynamic RAM. In each case, the address lines are reconfigured to accommodate the different number of rows and columns, and the Refresh Counter number sequence is changed to match the number of rows.

A RAM controller has essentially four functions: multiplex the RAM address, generate the RAM control signals, arbitrate memory access and refresh requests, and administer the refresh cycle. Figure 2-12 shows a simple block diagram of the controller. In the upper left corner is the address multiplexing section.  $AL_0-AL_6$  are the low-order or row address input bus lines, while  $AH_0-AH_6$  are the high-order or column address input bus lines. These are multiplexed and coupled to output bus lines  $OUT_0-OUT_6$  as described earlier. Notice that the Refresh Counter is also part of the row multiplex circuit. The counter generates a sequential series of 8-bit binary numbers starting with 0 and incrementing up to 127 (255 for 64K RAM). These form the row address during a refresh cycle. Naturally, during a refresh cycle, the multiplex circuit will only couple the Refresh Counter "row" address values to the output bus lines.

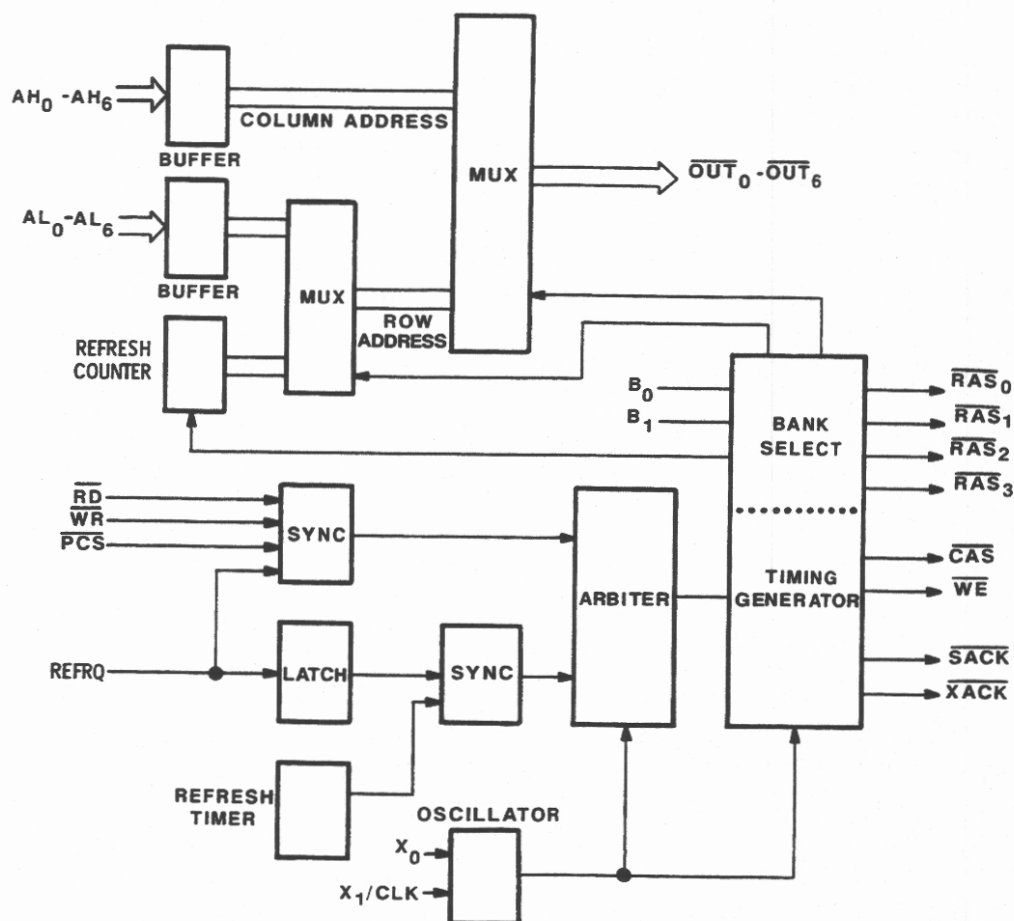


Figure 2-12

Simplified block diagram of the 8203 Dynamic RAM Controller.

Controller timing is handled by its internal oscillator. Provision is made for connecting either a crystal or a stable clock to the oscillator circuit. The oscillator frequency is approximately 20 MHz. Because the Controller timing circuit is not synchronized with the MPU clock, all of the control inputs must pass through synchronizer circuits. Once the timing of all the signals is synchronized, an Arbiter circuit determines the next operation to be performed by the Controller.

One of three operations can occur: a memory read, a memory write, or a memory refresh. The first two require that the Protected Chip Select (*PCS*) control line be low before either the *RD* or *WR* lines go low. This line is used to enable the memory read and write inputs. Then once a cycle begins, it will not abort, even if *PCS* goes inactive (high) before the cycle is completed. Thus for a read operation, *PCS* is pulled low, then *RD* is pulled low. The asynchronous read request is synchronized and sent to the Arbiter. If a refresh isn't pending, the read request will be sent to the Timing Generator. From here, the appropriate timing signals are generated to send the row and column addresses, and *RAS* and *CAS* to the RAM. The process is essentially the same for a write cycle. The only difference is that, in addition to the row and column addresses, and the *RAS* and *CAS* pulses, the *WE* line is pulled low to indicate a write operation.

While there is only one *CAS* signal, there are four different *RAS* signals. These are provided so the Controller can address four different banks of 16K RAM, for a total of 64K of dynamic RAM. The specific *RAS* signal is determined by Bank Select Inputs *B0* and *B1*. Normally, address line *A14* is connected to *B0* and address line *A15* is connected to *B1*. If you must access more than 64K of RAM, then you will need additional Dynamic RAM Controllers. Also, you will have to externally multiplex the high-order address lines, to come up with the appropriate inputs to *B0* and *B1* on each Controller.

It may seem odd that separate *RAS* signals are provided for each memory bank, while there is only one *CAS* signal. The answer is quite simple. As you may recall, *RAS* always initiates a memory address cycle. Until the memory row address generator is toggled, the memory column address generator can't toggle. Thus, only one *CAS* signal is needed to control all of the RAM.

Two other Controller outputs that haven't been mentioned are Transfer Acknowledge ( $\overline{XACK}$ ) and System Acknowledge ( $\overline{SACK}$ ). They provide a memory access feedback mechanism to the MPU.  $\overline{XACK}$  is a logic-low pulse that indicates data is valid during a read cycle, or data has been written during a write cycle. Figure 2-13 shows the timing relationship between  $\overline{RAS}$ ,  $\overline{CAS}$ , and  $\overline{XACK}$ . While  $\overline{XACK}$  pulses low to indicate the *end* of a memory access cycle,  $\overline{SACK}$  pulses low to indicate the *beginning*. It is also shown in Figure 2-13. Note that if a memory request is made during a refresh cycle,  $\overline{SACK}$  is delayed until  $\overline{XACK}$  goes active. Both the "normal" and "delayed" negative pulse transitions are shown in the figure.

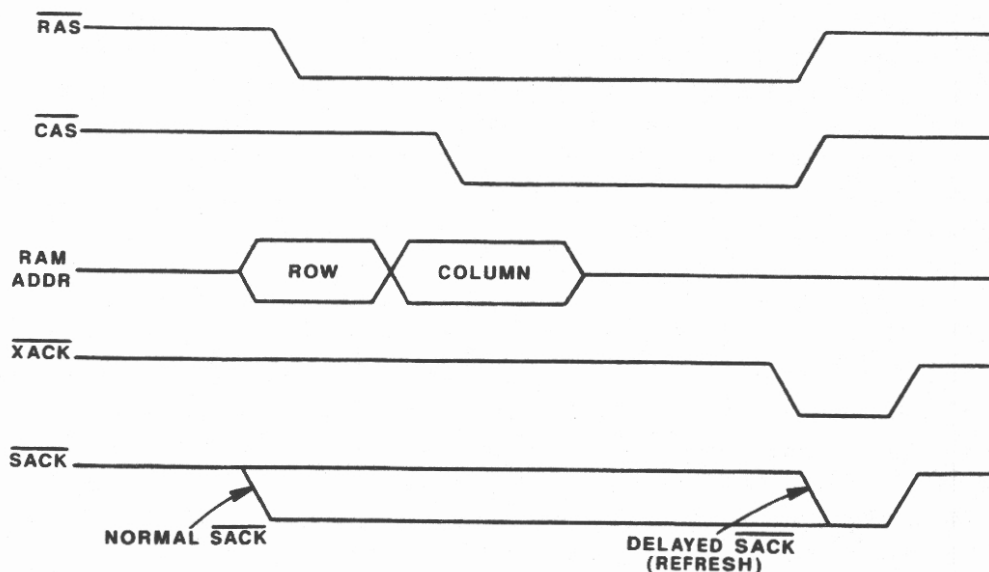


Figure 2-13

Dynamic RAM controller output signal timing.

Refresh can be handled in two different manners with the 8203, hidden and distributed. In hidden refresh, an external refresh signal is latched into the Controller, synchronized to the oscillator, and tested by the Arbiter. If the signal is properly timed, there won't be a memory cycle in progress, and refresh will be initiated immediately. If on the other hand, there is a memory cycle in progress, the Arbiter will ignore the refresh request. But because the request is latched, refresh will commence at the end of the current cycle. In addition, a simultaneous memory request and a refresh request will result in the memory request being honored first.

Distributed or internal refresh is generated by the Refresh Timer. The timer uses the 8203 clock to ensure that refresh occurs every 2 milliseconds. If the REFRQ line is inactive, the Refresh Timer will request a refresh cycle every 10-16 microseconds.

If a memory cycle is requested during a refresh cycle, the 8203 will set the internal delayed- $\overline{SACK}$  latch. When the memory cycle is eventually started, the 8203 will delay the active  $\overline{SACK}$  transition until  $\overline{XACK}$  goes active, as shown earlier in Figure 2-13. This delay is designed to compensate for any delays in the MPU's setup and hold times. The delayed- $\overline{SACK}$  latch is cleared after every memory cycle.

## Connecting Dynamic RAM To The MPU

Figure 2-14 is a simple diagram showing how the 8203 can be used to interface four banks of 16K RAM. The RAM has been arranged in this manner to reduce the complexity of the circuit. Still, it is easy to visualize that  $\overline{WE}$ ,  $\overline{CAS}$ , and  $\overline{RAS}$  are tied to each device in a bank, and that each  $D_{IN}$  and  $D_{OUT}$  line is tied to a separate data bus line. While all of the memory uses the  $\overline{WE}$  and  $\overline{CAS}$  signals, a separate  $\overline{RAS}$  signal is sent to each bank as described earlier.

Dynamic RAM timing is critical and often requires the MPU to "wait" until the RAM is ready. "Wait" generation in this example is controlled by the  $\overline{SACK}$  output. The signal is inverted and synchronized with the clock signal to control the  $\overline{HOLD}$  input to the MPU. As long as the  $\overline{HOLD}$  input remains high, the MPU knows that the RAM controller is busy. To guarantee a good data read cycle, the data coming from the RAM is latched. The  $\overline{XACK}$  pulse is used to latch, or strobe, the data into the Data Latch, and the inverted MPU  $\overline{RD}$  signal enables the output of the latch. It isn't necessary to latch data being written to RAM.

## 64K AND LARGER RAMs

The organization and operation of a typical 64K bit RAM is virtually identical to that of the 16K RAM. Most 64K RAMs are organized as 64K or 65,536 locations for one bit words. 16 bits are required to address 64K locations. However, only eight address lines are provided on the 64K RAM chip. The two 8-bit portions of the 16-bit address are multiplexed on these lines as they are in the 16K device. All other operations, read, write, and refresh are similar to the 16K device. Typical access times are 150 to 200 nanoseconds.

While the 16K, 64K and 256K dynamic RAMs are the most popular and widely used, a 1M bit RAM is also available. This RAM chip with its enormous storage capacity is not practical for some applications. However, it will find use in 16-bit microcomputers and large mainframe computers. Of course, the semiconductor manufacturers will continue to improve the technology and make even higher density RAMs available in the future.



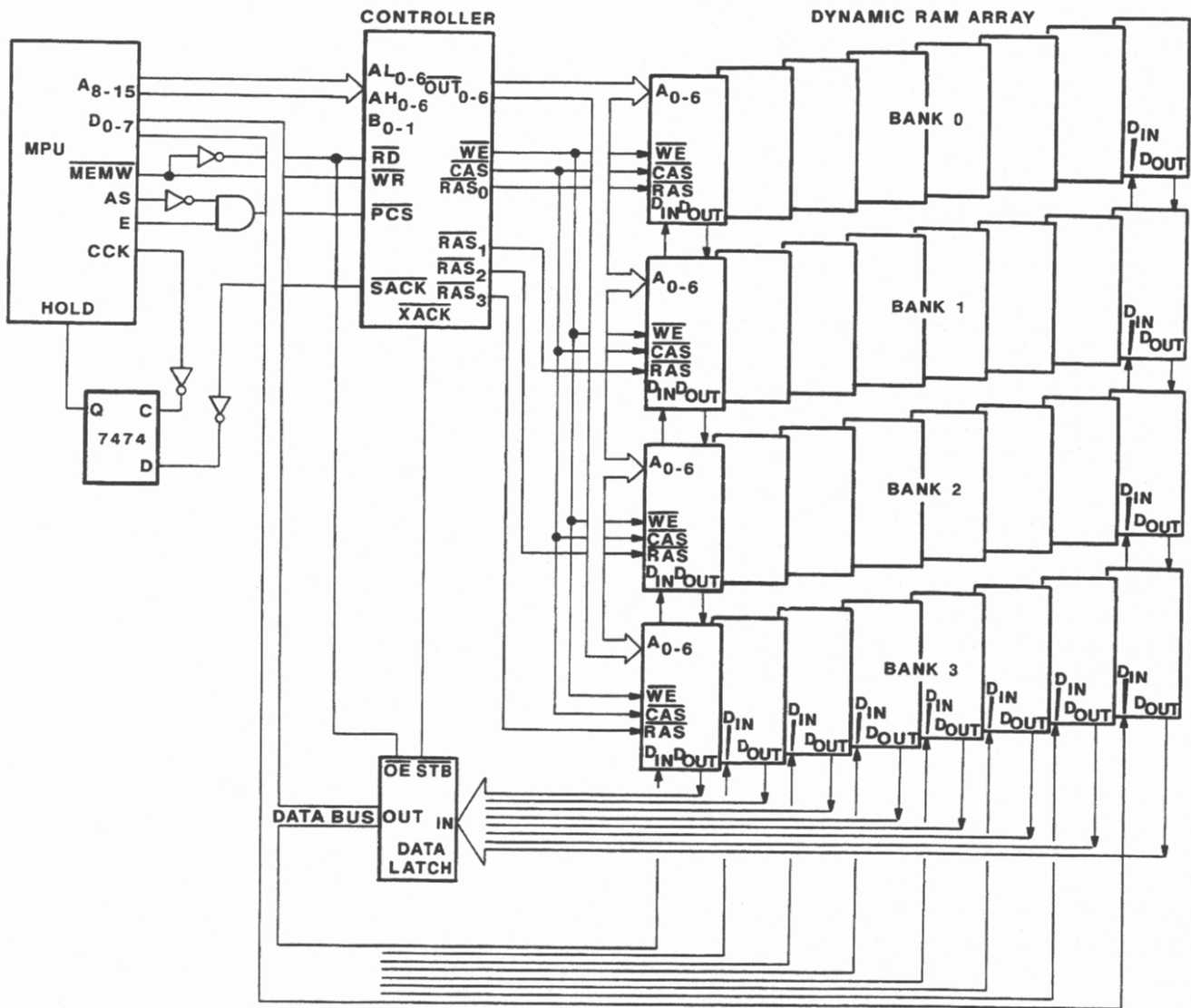


Figure 2-14

Interfacing four banks of 16K RAM with the 8203 Dynamic RAM Controller.

## Self-Test Review

1. The two basic types of RAM are static and \_\_\_\_\_.
2. Static RAM uses an integrated \_\_\_\_\_ for each of its storage cells.
3. A static RAM with a  $64 \times 64$  cell array can store up to \_\_\_\_\_ data bits.
4. Write Enable and Chip Select are two common static RAM control lines. What would be the status of RAM if  $\overline{WE}$  and  $\overline{CS}$  are high?
5. What is the biggest advantage of static RAM over dynamic RAM?
6. The storage cell in a dynamic RAM consists of a MOSFET and a \_\_\_\_\_.
7. As a general rule of thumb, you would use \_\_\_\_\_ RAM in a memory circuit that needs 32K of data storage.  
(static/dynamic)
8. When you access a row of dynamic storage cells, their data is \_\_\_\_\_ read by the column sense amplifier.
9. Generally speaking, dynamic RAM is \_\_\_\_\_-wide memory.
10. A  $16K \times 1$ -bit RAM contains a storage cell array of \_\_\_\_\_ rows and \_\_\_\_\_ columns.
11. The term  $\overline{RAS}$  stands for \_\_\_\_\_, while  $\overline{CAS}$  stands for \_\_\_\_\_.
12. Except for a refresh operation,  $\overline{CAS}$  always precedes  $\overline{RAS}$ . \_\_\_\_\_  
(True/False)
13. A 16K RAM must be refreshed at least every \_\_\_\_\_.
14. Two general types of refresh are distributed and \_\_\_\_\_.

15. If the memory controller refresh timer times out, it generates a \_\_\_\_\_ type of refresh.
16. The four basic functions of the memory controller are: \_\_\_\_\_ the RAM address, generate the RAM control signals, arbitrate memory access and refresh requests, and administer the refresh cycle.
17. The \_\_\_\_\_ line on the memory controller enables the memory read and write inputs.
18. Read cycle data valid is indicated by a logic-low pulse on the \_\_\_\_\_ output line.
19. When there is more than one bank of RAM, the memory controller uses the \_\_\_\_\_ signal to select one bank over another.

## Answers

1. The two basic types of RAM are static and dynamic.
2. Static RAM uses an integrated flip-flop for each of its storage cells.
3. A static RAM with a  $64 \times 64$  cell array can store up to 4K data bits.
4. If  $\overline{WE}$  and  $\overline{CS}$  are high, the RAM is disabled and its data lines are floating.
5. The biggest advantage of static RAM over dynamic RAM is that static RAM doesn't need to be refreshed periodically.
6. The storage cell in a dynamic RAM consists of a MOSFET and a capacitor.
7. As a general rule of thumb, you would use dynamic RAM in a memory circuit that needs 32K of data storage.
8. When you access a row of dynamic storage cells, their data is destructively read by the column sense amplifier.
9. Generally speaking, dynamic RAM is a bit-wide memory.
10. A  $16K \times 1$ -bit RAM contains a storage cell array of 128 rows and 128 columns.
11. The term  $\overline{RAS}$  stands for Row Address Strobe, while  $\overline{CAS}$  stands for Column Address Strobe.
12. False.  $\overline{RAS}$  always precedes  $\overline{CAS}$  regardless of operation.
13. A 16K RAM must be refreshed at least every 2 milliseconds.
14. Two general types of refresh are distributed and hidden.
15. If the memory controller refresh timer times-out, it generates a distributed type of refresh.

16. The four basic functions of the memory controller are: **multiplex** the RAM address, generate the RAM control signals, arbitrate memory access and refresh requests, and administer the refresh cycle.
17. The  $\overline{PCS}$  line on the memory controller enables the memory read and write inputs.
18. Read cycle data valid is indicated by a logic-low pulse on the  $\overline{XACK}$  output line.
19. When there is more than one bank of RAM, the memory controller uses the  $\overline{RAS}$  signal to select one bank over another.

---

## **EPROM PROGRAMMING AND INTERFACING**

The EPROM (Erasable Programmable Read Only Memory) is a user-programmable and erasable device. EPROMs are programmed using a PROM programming, or burning, unit and erased by exposing them to high intensity ultraviolet light. Once programmed, an EPROM acts like a ROM and is non-volatile. EPROMs have all the desirable features required for early system development. They can be erased and reprogrammed many times to fix program bugs that often occur during system development.

## A Typical EPROM Architecture

Although many (larger) versions are available, we'll use a  $2K \times 8$  bit EPROM as an example. Figure 2-15 shows a block diagram of such an EPROM.

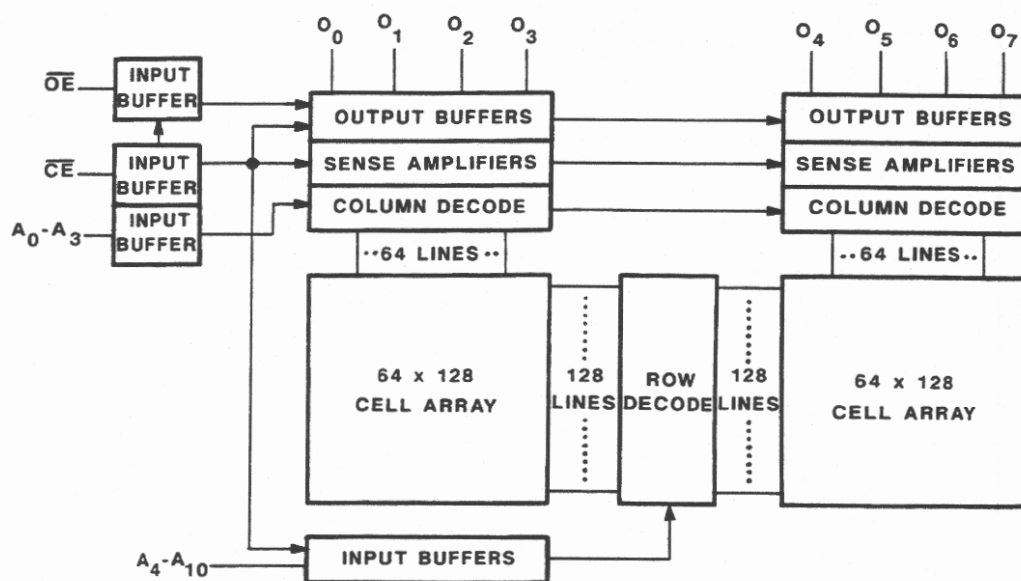


Figure 2-15  
Block diagram of a  $2K \times 8$  EPROM.

This is a 16K-bit memory configured as 2K bytes. Internally, these 16K bits are arranged as two  $64 \times 128$  storage cell arrays, with the row decoder situated between the arrays. The decoder was physically placed between the two arrays to increase the speed of cell selection. High order address lines  $A4$  through  $A10$  determine which of the 128 rows is accessed. Each of these address lines, as well as the other address lines and the control lines, are buffered, with buffer operation controlled by the Chip Enable line ( $\overline{CE}$ ).

Once a row has been accessed, the column decoder must select not just one, but *eight* different storage cells from the active row. This is because we are now dealing with *byte-wide* memory. The task really isn't that complex, however, when you consider that the column decoder is divided into eight identical sections. Each section is then called upon to supply one *bit* of data to an output. Thus, while the EPROM is physically divided into two  $64 \times 128$  arrays of storage cells, it is electrically subdivided into eight  $16 \times 128$  arrays, each supplying data to one output line. Figure 2-16 shows one of these smaller arrays.

In this small cell array, a portion of the storage MOSFETs are shown, with the rows labeled  $X0$  through  $X127$  and the columns labeled  $Y0$  through  $Y15$ . High order address lines  $A4$  through  $A10$  decode the row, while low order address lines  $A0$  through  $A3$  decode the column. Therefore, if the memory address is 00000000000, the MOSFET at array location  $X0, Y0$  will be selected. By the same token, if the address is 11111111111, the selected MOSFET is at location  $X127, Y15$ . Because each  $16 \times 128$  cell array is identical to the next, when you address MOSFET  $X127, Y15$  in one array, you are also addressing MOSFET  $X127, Y15$  in the other seven arrays.

Once a storage cell (MOSFET) is selected, its logic value is coupled through the sense amplifier to the data output buffer. The buffer will then hold that logic value until a new value is accessed by the address lines. The Output Enable ( $\overline{OE}$ ) line determines when the buffer output is active.

The logic level of an EPROM's storage cell is determined by the condition of its *floating gate*. The non-programmed state of the floating gate storage cells is usually a logic 1 state, while the programmed state is a logic 0. Thus, when programming an EPROM, you are actually programming 0's into the device.



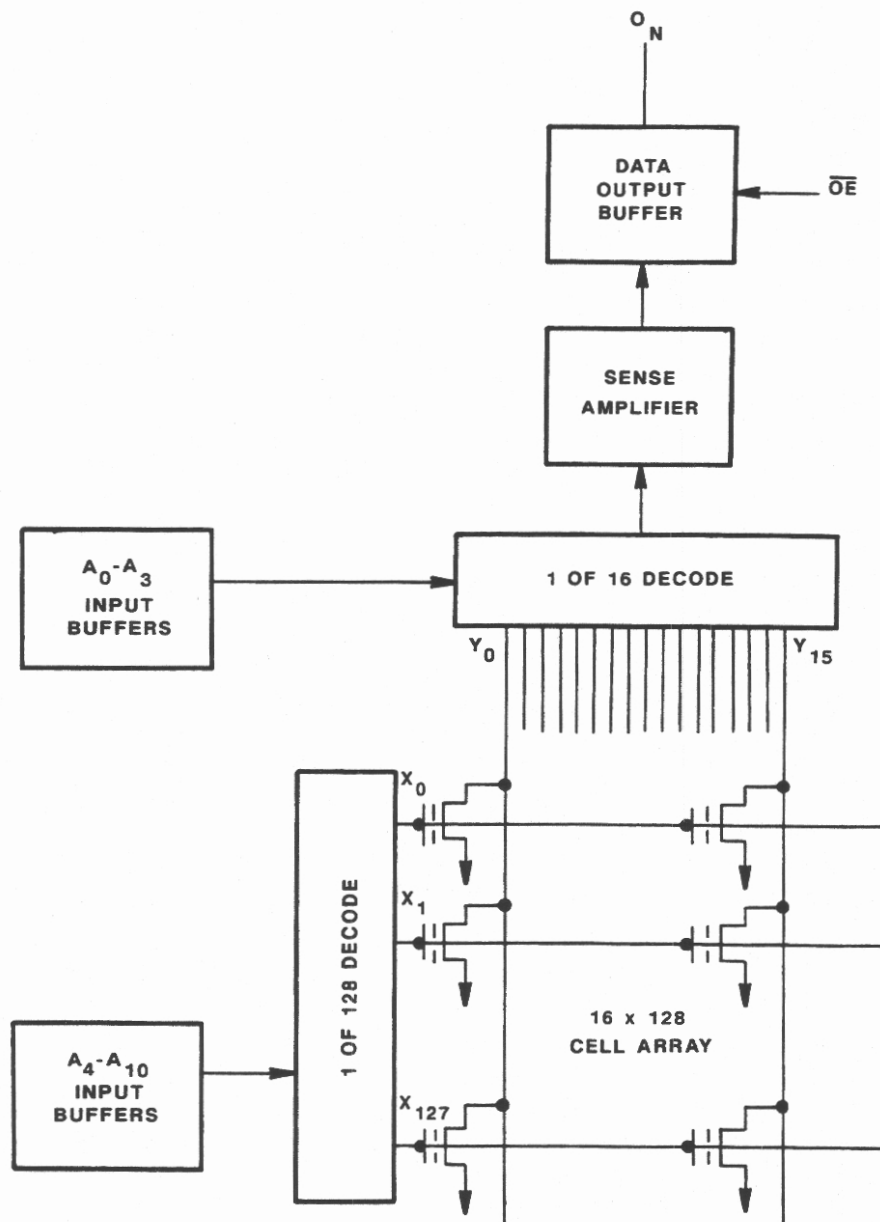


Figure 2-16  
One of eight single-bit data cell arrays.

To reduce the complexity of the block diagram, the EPROM programming circuit in Figure 2-15 wasn't shown. Figure 2-17, however, does show a portion of the circuit connected to our small cell array. It consists of an input buffer connected directly to the column decoder, and two programming lines.

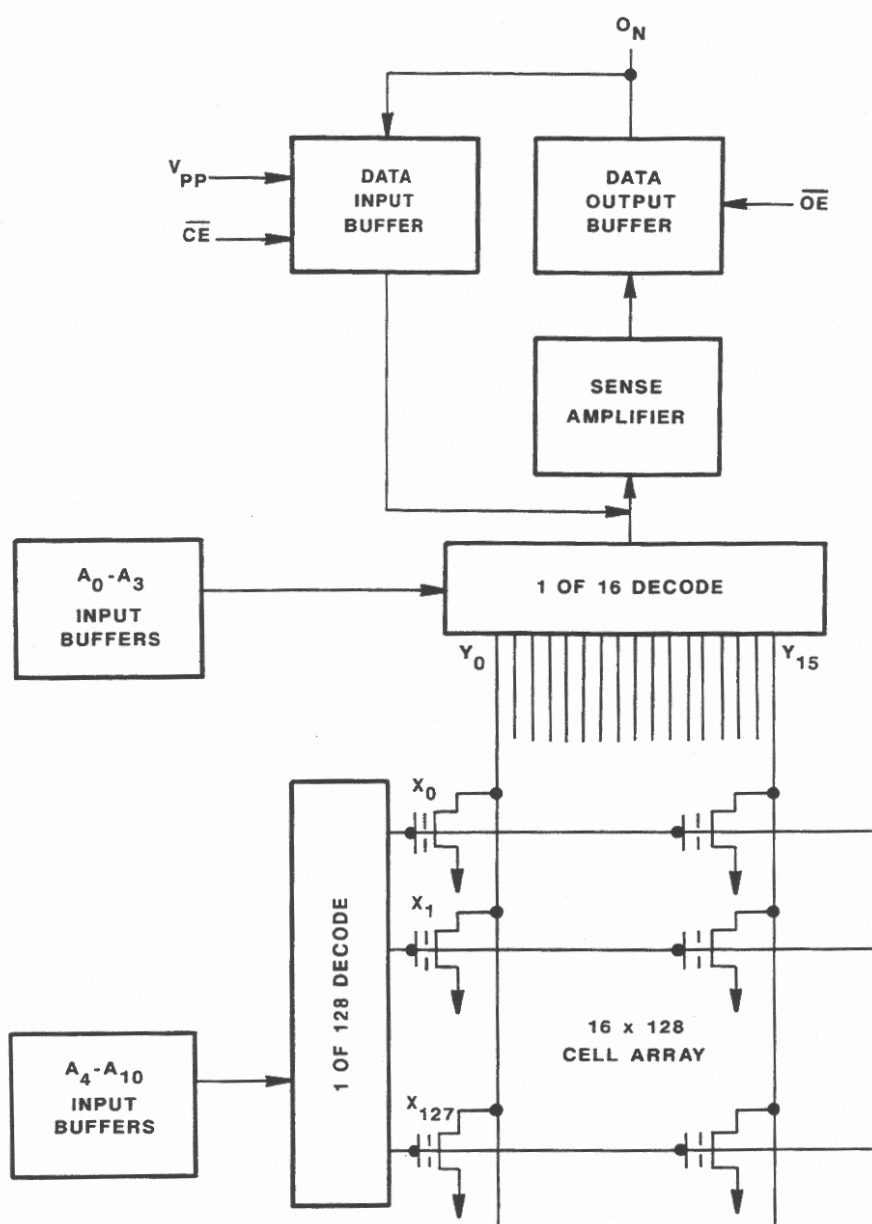


Figure 2-17

One of eight single-bit data cell arrays, with input data buffering added.

## Programming the EPROM

During the programming operation, the data output lines become data input lines. Programming control line  $V_{pp}$  which is normally held at +5 volts, is pulled to between +12 and +25 volts depending on the particular device used. This deselects the output buffer and selects the input buffer. It also supplies the necessary potential to charge a MOSFET floating gate.

To program a cell, the address of the cell is decoded and the appropriate data is presented to the input buffer. While this is happening, the Output Enable line is held high (inactive) and the Chip Enable ( $\overline{CE}$ ) line is held low (active). The address, data, and Chip Enable signals must be stable for at least 2 microseconds. After that delay period, the Chip Enable line is pulsed high for a period of 50 milliseconds. It is during this period that the selected cell is programmed. After the Chip Enable line goes low, the address, data, and Chip Enable lines must again remain stable for 2 microseconds. This completes the programming cycle. If you want to continue programming, you can change the address and data values immediately after the second delay; and then after another 2 microsecond delay, pulse the Chip Enable line. In this manner, you can program the entire EPROM in approximately 100 seconds. Returning the  $V_{pp}$  programming control line voltage to +5 volts causes the EPROM to again operate as Read-Only Memory.

One last item that wasn't covered earlier is device and system power. Every electrical device in a microcomputer consumes energy. This dictates the capacity of the system power supply, and the heat generated inside the microcomputer housing. Naturally, the ideal system would draw no power, and generate no heat. This, of course, is not possible with today's technology. However, many of the newer devices do have a power-down mode that helps conserve energy and reduce heat.

When the EPROM is active, that is, when the Chip Enable line is low, it is drawing approximately 500 milliwatts of power. However, when the Chip Enable line is high, the EPROM is essentially inactive. In this state, its internal functions are reduced to a bare minimum, and it only draws about 125 milliwatts of power, a reduction of 75%. Now consider the fact that when ROM is being accessed, RAM is inactive, and so on. It becomes apparent that if you design a system around devices that have a power-down mode, you can reduce the *worst case* power supply requirements by a substantial margin.

## Self-Test Review

20. The EPROM is user-programmable and \_\_\_\_\_.
21. The logic level of an EPROM's storage cell is determined by the condition of its \_\_\_\_\_ gate.
22. The nonprogrammed state of the storage cells in an EPROM is a logic \_\_\_\_\_.
23. When the floating gate in an EPROM storage cell is charged, the cell logic level is \_\_\_\_\_.
24. A 2K x 8 EPROM is a \_\_\_\_\_ K-bit memory device.
25. Explain how an EPROM is programmed.

## Answers

20. The EPROM is user-programmable and erasable.
21. The logic level of an EPROM's storage cell is determined by the condition of its floating gate.
22. The nonprogrammed state of the storage cells in an EPROM is a logic one.
23. When the floating gate in an EPROM storage cell is charged, the cell logic level is low.
24. A 2K x 8 bit EPROM is a 16 K-bit memory device.
25. When an EPROM is programmed, the data output lines become data input lines. To program a one byte cell, the address of the cell is decoded and the appropriate data is presented to the input buffer. After a short delay, the Chip Enable line is pulsed high for a period of 50 milliseconds. This gates a +25 volt programming pulse to the cell. This process is then repeated for each cell address until the desired number of cells are programmed.

## THE ETW-3800 (8085) TRAINER MEMORY MAP

Figure 2-18 shows the memory map for the 8085 version of the ETW-3800 Trainer. As you can see, the system ROM is located in the lower 28K address space, from address 0000H\* through 5FFF. The system ROM contains the ROM monitor program which is responsible for controlling the various Trainer functions.

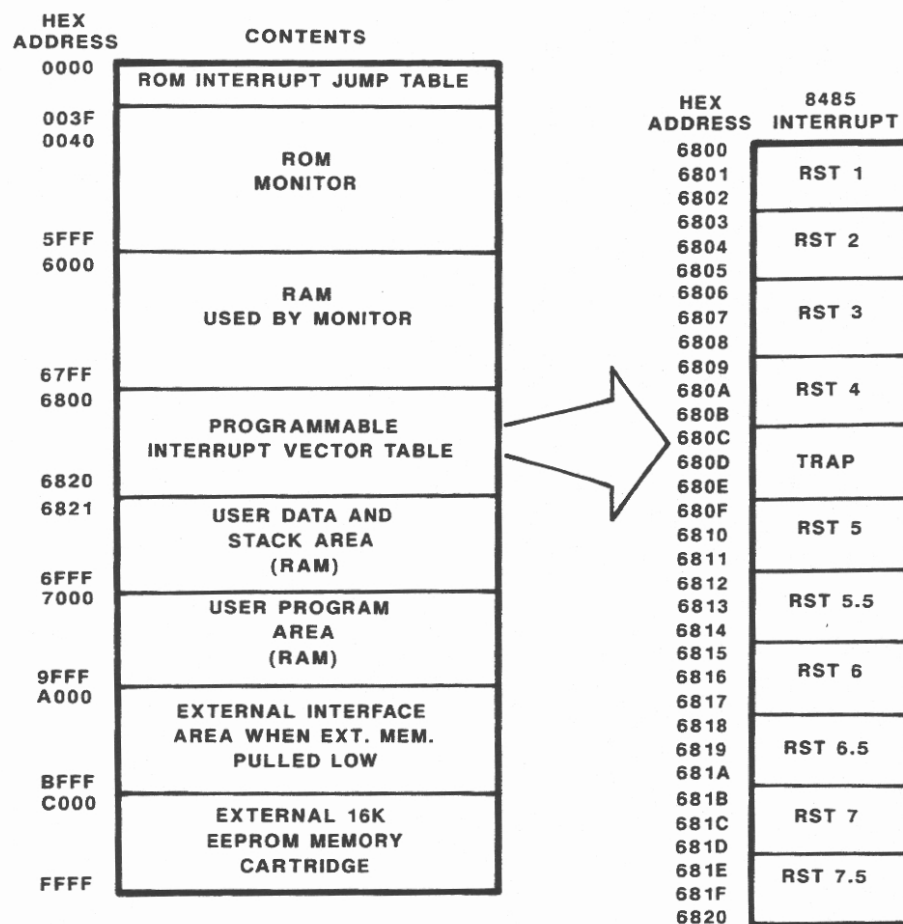


Figure 2-18  
The ETW-3800 Trainer Memory Map (8085 Version).

**NOTE** — When specifying an address, Hexidecimal may be indicated by using *either* a subscript of 16, e.g., 1000<sub>16</sub>, or a capital letter "H" immediately following the address, e.g., 1000H.

Notice that the 8085 interrupt jump table is located at the beginning of this region. This is where the 8085 interrupt vectors are located. When an interrupt occurs, the 8085 MPU goes to this region to obtain the respective interrupt vector. Since the vectors are in ROM, they are fixed by the Trainer designer and cannot be altered. However, a given vector directs the MPU to the programmable interrupt vector table where you can insert your own interrupt vectors. This programmable interrupt vector table begins at address 6800H and is shown in the exploded view of Figure 2-18. Here is where you can insert JUMP instructions to use the respective 8085 interrupts. The JUMP instruction that you insert must jump to an address in the user program area where you have located the respective interrupt service routine.

The RAM used by the ROM monitor program is located at addresses 6000H through 67FFH. Although this area is RAM, you should not use it for your programs or data because the ROM monitor will most likely write over and destroy any information in this region.

The user RAM area is located from address 7000H through 9FFFH.

This is where you must locate any programs you have written. In addition, addresses 6821H through 6FFFH are available for your program data and stacks.

Addresses A000H through BFFFH are available for external interfacing when the ENEXM control line on the Trainer is pulled low. Otherwise, this area provides an additional 8K bytes of user RAM.

Finally, the 16K external EEPROM cartridge is located from addresses C000H through FFFFH. You can use this area to save your programs on a semi-permanent basis if you have the EEPROM cartridge.

## I/O Memory Map

As you know, the 8085 is capable of communicating with I/O devices using either direct I/O or memory-mapped I/O. The ETW-3800 Trainer employs direct I/O to communicate with the various Trainer I/O devices. Figure 2-19 shows the Trainer I/O port address assignments. You will become more familiar with many of these as you use them in the course experiments.

PORT ADDRESS	I/O DEVICES
00-0F	KEYBOARD COLUMN 1
10-1F	KEYBOARD COLUMN 2
20-2F	KEYBOARD COLUMN 3
30-3F	LED CONTROL, AID CONTROL
41, 45, 48, 4C	TIMER REGISTER 1 (ACCESS VIA 41H)
41, 45, 49, 4D	TIMER REGISTER 2 (ACCESS VIA 41H)
42, 46, 4A, 4E	TIMER REGISTER 3 (ACCESS VIA 43H)
43, 47, 4B, 4F	TIMER REGISTER 4 (ACCESS VIA 43H)
50-5F	GENERAL I/O 1
60-6F	GENERAL I/O 2
70	RAM PROTECT SELECT
80-83	A/B INPUT, D/A OUTPUT
90-93	INPUT PORT
A0-A3	OUTPUT PORT
B0, B2	LCD DATA
B1, B3	LCD CONTROL
B4-FF	UNUSED

Figure 2-19  
The ETW-3800 I/O Memory Map (8085 Version).



## UNIT SUMMARY

Random access memory, or RAM, is any memory that lets you directly access any storage cell locations. However, by convention read/write memory is considered to be RAM, although read only memory is also random access memory. There are two basic types of RAM, static and dynamic. Static RAM employs flip-flops to store binary data. Since data is latched in these flip-flops, no memory refresh is required. Dynamic RAM, on the other hand, requires refresh since each bit of data is stored as a charge level in a capacitor.

Interfacing to static RAM is simple. You must design an address decoder to decode the designated RAM addresses and enable the respective RAM devices. The read/write line from the MPU signals the RAM devices to perform either a read or a write operation.

Interfacing to dynamic RAM is not so simple because of the refresh requirement. Many dynamic RAM devices require refreshing at least once every two milliseconds. There are two general methods employed for refreshing dynamic RAM: distributed refresh, and hidden refresh. It is often desirable to employ a single-chip dynamic refresh controller to handle the refreshing task because of its complexity.

An EPROM is a user-programmable and erasable ROM device. EPROMs are often used as ROMs during early system development when program bugs are common. An EPROM is programmed by gating a relatively high voltage to the EPROM storage cells. This high voltage causes a charge to be stored on a floating gate device within the cell. An EPROM is erased by exposing it to high intensity ultraviolet light. The light energy allows the stored charges to dissipate within the memory cells. EPROMs can be erased and reprogrammed many hundreds of times without damage.

## *Unit 3*

# **PROGRAMMABLE I/O DEVICES**

## CONTENTS

Introduction .....	3-3
Unit Objectives .....	3-4
Programmable Parallel I/O Ports .....	3-5
Fundamental Concepts .....	3-5
The Programmable Parallel I/O Section .....	3-7
Internal Registers .....	3-8
Interfacing and Addressing .....	3-13
Initialization .....	3-15
Self-Test Review .....	3-17
Answers .....	3-18
Using a Programmable Parallel I/O Port .....	3-20
Driving 7-Segment Displays .....	3-20
Decoding Keyboards .....	3-24
Decoding a Switch Matrix .....	3-26
Self-Test Review .....	3-28
Answers .....	3-29
I/O Control Techniques .....	3-30
Interrupt Control of I/O Operations .....	3-30
Handshaking With the MUART .....	3-32
Input Handshake .....	3-32
Output Handshake .....	3-34
Configuring the MUART for Handshake Operations .....	3-36
Interrupt Control Using the MUART .....	3-37
Using the 8085 INTR Interrupt Input .....	3-38
Using the RST 7.5, RST 6.5, RST 5.5, or TRAP	
Interrupt Inputs .....	3-40
Configuring the MUART for Interrupt Servicing .....	3-43
Interrupt Enabling Within the MUART .....	3-44
The P17 and EXTINT Interrupts .....	3-45
Polled Control of I/O Operations .....	3-46
Self-Test Review .....	3-49
Answers .....	3-50
Unit Summary .....	3-51

## Unit 3

# PROGRAMMABLE I/O DEVICES

## INTRODUCTION

In this unit, you will be introduced to the programmable interface devices that provide parallel I/O, serial I/O, and timing functions. You have already learned that interfacing to a microprocessor requires elements of address decoding, three state buffering, and latching. Up to this point, you have used discrete ICs to provide these functions. Now you are about to learn how a single programmable device can be employed to provide all of these interfacing requirements.

An important feature of many programmable I/O devices is that they can be programmed, or *configured*, by the MPU to perform a variety of interfacing tasks. In particular, you will see how a single device, called the 8256 MUART, can be programmed to provide parallel I/O, serial I/O, and timer operations. As a result, you will find that the MUART is a very flexible device.

This unit will expose you to the various features of the MUART and show you how the MUART is used to provide programmable parallel I/O. Then, in Units 4 and 5 you will learn how the MUART is used to provide serial I/O and programmable timer operations, respectively. Now, let's learn more about this interesting device.

## UNIT OBJECTIVES

1. Explain the function and use of programmable I/O devices.
2. State why programmable I/O devices are superior to combinational logic for interfacing to external peripheral devices.
3. List the internal functions of the 8256 MUART programmable I/O device.
4. Explain how the 8256 MUART provides programmable parallel I/O and parallel handshaking.
5. Write a simple initialization routine to configure the MUART for parallel I/O.
6. Explain how the MUART can be used to multiplex displays and decode keyboards.
7. Explain the functions of the MUART internal registers that are used in connection with parallel I/O and handshaking.
8. Sketch a circuit diagram showing the required connections between the 8085 MPU and the 8256 MUART.
9. Describe the interrupt handling process of the MUART for any of the 8085 interrupt input lines.
10. List the interrupt sources of the MUART.
11. Explain how the MUART is used for both interrupt and programmed control of I/O operations.

## PROGRAMMABLE PARALLEL I/O PORTS

Most microprocessors have a family of support chips that are used to simplify the problem of interfacing with the outside world. Traditionally, these support chips include programmable ICs that provide parallel I/O, serial I/O, and timer functions, among others. With earlier large scale integrated (LSI) circuit technology these functions were integrated onto separate ICs. Today, with very large scale integrated (VLSI) circuit technology, many IC manufacturers are integrating several programmable functions on a single IC. In particular, Intel\* has integrated programmable parallel I/O, serial I/O, and timer functions onto a single IC called the *8256 Multifunctional Universal Asynchronous Receiver Transmitter*, or *MUART*. The 8256 MUART design allows it to be connected directly to Intel's 8085, 8086, 8088, 80186, and 80188 MPUs.

### Fundamental Concepts

The purpose of any programmable I/O device is to simplify the problem of interfacing the MPU to external devices. Of course, any device can be interfaced with the MPU using conventional combinational logic. However, the conventional logic approach generally requires many ICs. This defeats one of the prime advantages of the microprocessor - a simple straightforward design requiring few ICs. The advantage of a programmable I/O device like the MUART is that one or two ICs can do all the interfacing.

Programmable devices like the MUART are superior to conventional logic in another way. They are extremely flexible because they are programmable. That is, their configuration can be changed from one moment to the next by the program being executed. For instance, an output port can be changed to an input port by software in the middle of a program. Try doing that with combinational logic! Furthermore, because these programmable ICs can do most routine peripheral control tasks, the MPU is freed to handle more important system related tasks.

\* Intel is a trademark of Intel Corporation.

The MUART acts as a buffer between the MPU and peripheral devices as illustrated in Figure 3-1. Here, you see that the MUART is the interface between the MPU and both serial and parallel peripheral devices. It is connected directly to the MPU address, data, and control buses on one side to allow communication with serial and parallel I/O devices on the other side. In other words, you could say that the MUART is *the* interface between the MPU and a peripheral device.

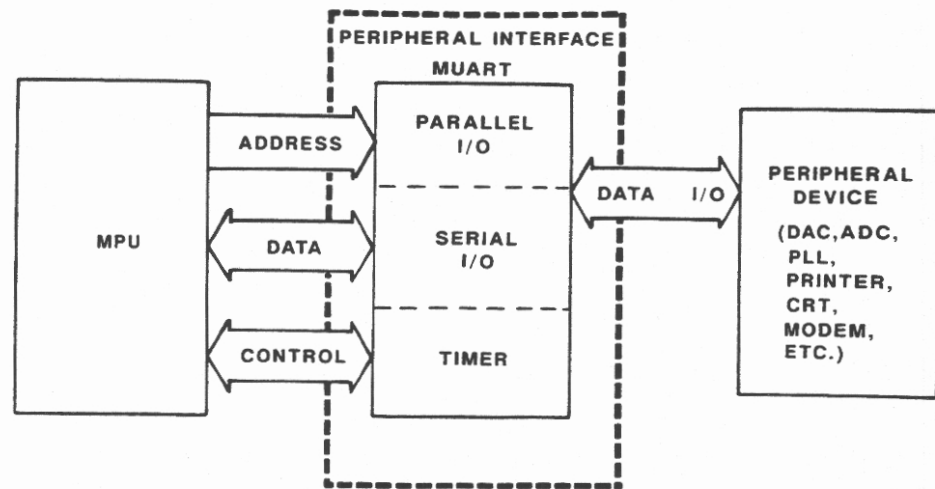


Figure 3-1

The MUART is the interface between the MPU and peripheral device.

In application circuits, the peripheral devices might include a keyboard, display, printer, MODEM, D/A converter, A/D converter, stepper motor, etc. The parallel I/O portion of the MUART communicates with parallel I/O devices, while the serial I/O portion of the MUART communicates with serial I/O devices. In addition to parallel and serial interfacing, the MUART has a programmable timer section that is capable of generating output signals as well as counting and measuring input signals.

In the remainder of this unit, we will focus on the parallel section of the MUART. Then, in Units 4 and 5 we will cover the serial and timer sections of the MUART, respectively.

## The Programmable Parallel I/O Section

The parallel I/O section of the MUART can be functionally divided into two sides, the MPU side and a peripheral side as illustrated in Figure 3-2.

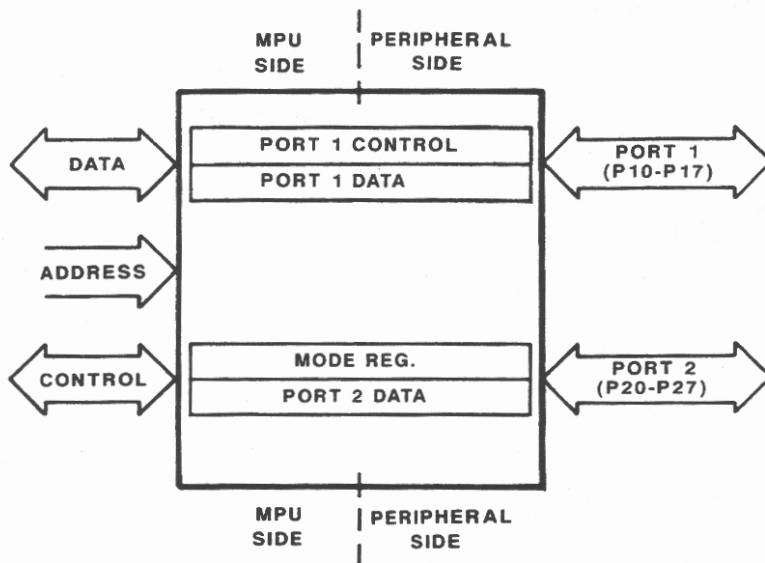


Figure 3-2

The parallel I/O section of the MUART.

The MPU side includes the data, address, and control lines which interface to the MPU data, address, and control buses. The peripheral side contains two 8-bit I/O ports, called Port 1 and Port 2. The eight Port 1 lines are labeled *P10* through *P17*, while the Port 2 lines are labeled *P20* through *P27*.

Port 1 is bit programmable, meaning that the eight bits of Port 1 can be individually programmed as input or output. Port 2 is *nibble* (4 bit) programmable, meaning that the lower four bits or the upper four bits can be collectively programmed for input or output.

Each port has an associated latch and three-state driver. When the port lines are programmed for output, data written to the port is latched and appears on the respective pins. Thus, any data written to the output port lines remains latched and unchanged unless new data is written to those port lines. When the port lines are programmed for input, reading the port gates the logic states at the port pins onto the MPU data bus via a three-state buffer action.



Writing to input port lines has no effect on the associated pins, but the data will be stored in the port latch and appear on the pins if the port is changed to an output port at a later time.

## INTERNAL REGISTERS

The internal registers associated with the parallel I/O section of the MUART are also shown in Figure 3-2. Here you see a *Mode Register*, a *Port 1 Control Register*, and two *Data Registers*. The Mode register is used to control the operation of Port 2. From Figure 3-3 you see that it is an 8-bit register; however, only the lower three bits are associated with Port 2. The upper five bits are used to control the programmable timer section of the MUART and will be discussed in Unit 5.

PORT ADDRESS

X3H

P2C2

P2C1

P2C0

PORT 2 CONTROL

MODE REGISTER

				DIRECTION	
P2C2	P2C1	P2C0	MODE	UPPER	LOWER
0	0	0	NIBBLE	INPUT	INPUT
0	0	1	NIBBLE	INPUT	OUTPUT
0	1	0	NIBBLE	OUTPUT	INPUT
0	1	1	NIBBLE	OUTPUT	OUTPUT
1	0	0	BYTE HANDSHAKE	INPUT	
1	0	1	BYTE HANDSHAKE	OUTPUT	
1	1	0	N/A	N/A	
1	1	1	TEST		

**Figure 3-3**

Port 2 control using the Mode Register.

The logic written to the lower three bits of the Mode register determines the direction of the Port 2 I/O lines. Notice that the port can be configured as an 8-bit input port, an 8-bit output port, or a nibble combination of input or output. For instance, to make Port 2 an 8-bit output port, you simply write 0 1 1 to the lower three bits of the Mode register. To make the upper half of Port 2 (P27-P24) input and the lower half of Port 2 (P23-P20) output you must write 0 0 1 to the lower three bits of the Mode register. That's all there is to it!

In addition to selecting the direction of the Port 2 I/O lines, you see from Figure 3-3 that you can also select an input byte handshake or an output byte handshake mode of operation. These two modes are selected by writing 1 0 0 or 1 0 1, respectively, to the lower three bits of the mode register. Handshaking allows for synchronization of data transfer between the MPU and peripheral device and will be discussed in detail shortly.

The Port 1 Control register shown in Figure 3-4 is used to select the direction of the individual Port 1 I/O lines.

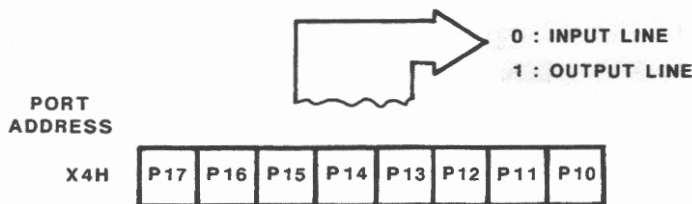


Figure 3-4

The Port 1 Control Register controls the direction of the Port 1 I/O lines.

Writing a logic 1 to a given bit configures the corresponding Port 1 I/O line for output. Writing a logic 0 to a bit configures the corresponding I/O line for input. Thus, to configure Port 1 as an 8-bit output port you must write the value FFH to the Port 1 Control Register, while to configure Port 1 as an 8-bit input port you must write the value 00H to the Port 1 Control Register. How will Port 1 be configured by writing the value 55H to the Control Register?

Finally, the two data registers are used to communicate data through the two ports. The data registers act as 3-state buffers for input operations, and latches for output operations. You can view each of the eight bits within a given data register as being connected to one of the respective port lines on the MUART chip as shown in Figure 3-5.

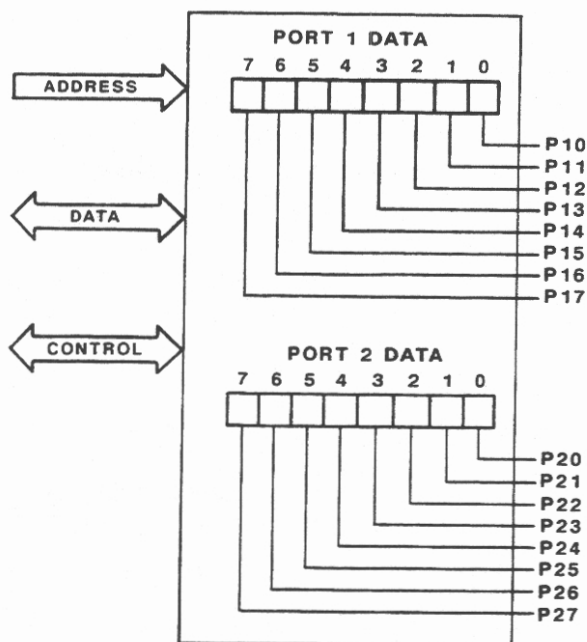


Figure 3-5

Port line data flow through the respective Data Register bits.

To output data to the lines configured for output, you simply write to the respective port data register. Likewise, to input data from those lines that have been configured for input, you simply read the respective data register.

To illustrate a typical configuration, Figure 3-6 shows Port 1 configured as an 8-bit input port and Port 2 configured as an 8-bit output port.

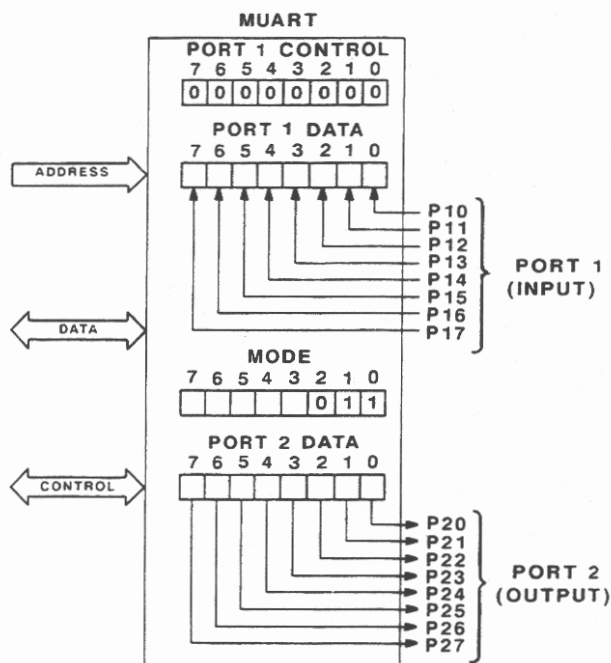


Figure 3-6  
Port 1 configured for input and Port 2 for output.

Notice that the Port 1 Control Register contains all 0s to configure Port 1 for all input, while the Mode Register contains the value 0 1 1 in its lower three bits to configure Port 2 for all input. Another configuration is shown in Figure 3-7. Here, Port 1 is configured for input as before, but Port 2 is configured for half input an half output via the Mode Register.

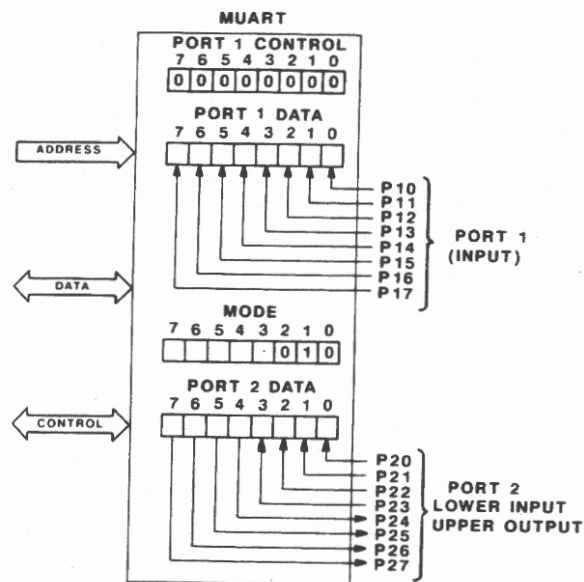


Figure 3-7

Various input/output combinations are possible through program control.

## Interfacing and Addressing

Because the 8256 MUART was designed to be directly compatible with the Intel\* family of MPUs, interfacing it to the 8085 MPU is straightforward. The circuit diagram in Figure 3-8 shows how it's done.

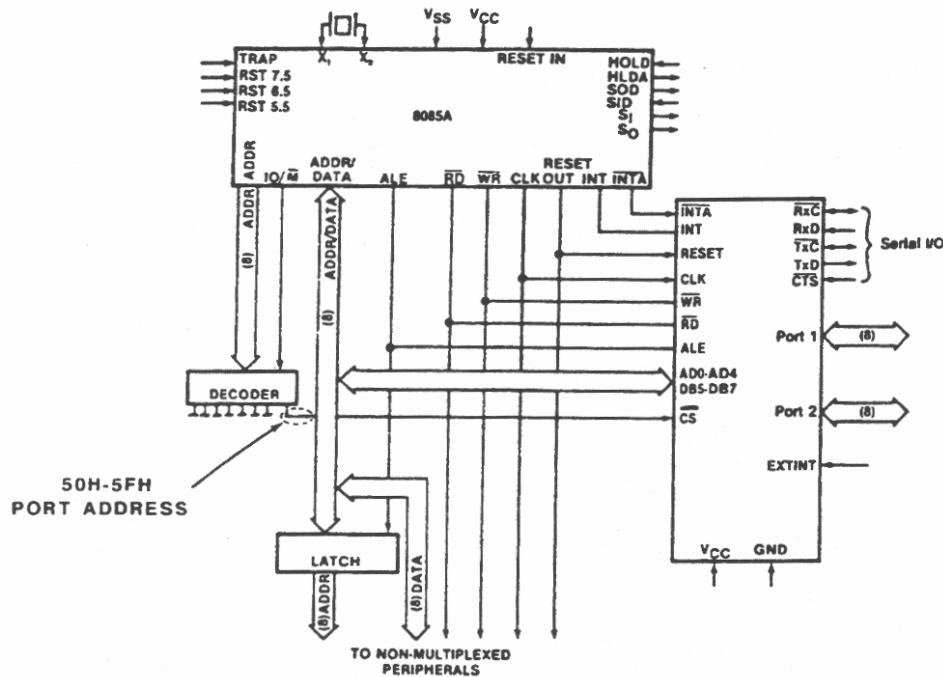


Figure 3-8

The 8085 MPU/8256 MUART interface connections.

As you can see, the MUART is designed to be connected directly to the 8085 multiplexed address/data bus. In fact, if you demultiplex the bus for other peripheral devices in the system as discussed in Unit 1, the MUART must be connected on the microprocessor side of the demultiplexer. The MUART internally demultiplexes the address/data bus information using the 8085 ALE line. Notice that ALE is connected directly to the MUART. In addition, the 8085  $\overline{INTA}$ ,  $\overline{INT}$ ,  $\overline{RESET}$ ,  $\overline{CLR}$ ,  $\overline{WR}$ , and  $\overline{RD}$  lines will connect directly to corresponding lines on the MUART.

Finally, the MUART is enabled via a single chip select ( $\overline{CS}$ ) line. As a result, there is no internal decoding provided within the MUART except for internal register decoding. You must use an external address decoder to decode the upper eight MPU address lines and enable the MUART chip select line. Here, the upper eight MPU address lines are decoded to generate a chip select signal when any port address within the range of 50H to 5FH is generated via an IN or OUT instruction. Consequently, the MUART is interfaced to the MPU for direct I/O.

\*Intel is a trademark of Intel Corporation.

Once the MUART is enabled via its chip select ( $\overline{CS}$ ) line, its internal registers can be accessed for initialization and operation of the device. There are sixteen 8-bit read/write register pairs within the MUART as shown in Figure 3-9.

### INTERNAL MUART REGISTERS

PORT ADDRESS	READ ONLY	WRITE ONLY
X0H	<div> <div>L1 L0 S1 S0 BRKI BITI 8086 FRQ</div> <div>Command 1</div> </div>	<div> <div>L1 L0 S1 S0 BRKI BITI 8086 FRQ</div> <div>Command 1</div> </div>
X1H	<div> <div>PEN EP C1 C0 B3 B2 B1 B0</div> <div>Command 2</div> </div>	<div> <div>PEN EP C1 C0 B3 B2 B1 B0</div> <div>Command 2</div> </div>
X2H	<div> <div>0 RxE IAE NIE 0 SBRK TBRK 0</div> <div>Command 3</div> </div>	<div> <div>SET RxE IAE NIE END SBRK TBRK RST</div> <div>Command 3</div> </div>
X3H	<div> <div>T35 T24 T5C CT3 CT2 P2C2 P2C1 P2C0</div> <div>Mode</div> </div>	<div> <div>T35 T24 T5C CT3 CT2 P2C2 P2C1 P2C0</div> <div>Mode</div> </div>
X4H	<div> <div>P17 P16 P15 P14 P13 P12 P11 P10</div> <div>Port 1 Control</div> </div>	<div> <div>P17 P16 P15 P14 P13 P12 P11 P10</div> <div>Port 1 Control</div> </div>
X5H	<div> <div>L7 L6 L5 L4 L3 L2 L1 L0</div> <div>Interrupt Enable</div> </div>	<div> <div>L7 L6 L5 L4 L3 L2 L1 L0</div> <div>Set Interrupts</div> </div>
X6H	<div> <div>D7 D6 D5 D4 D3 D2 D1 D0</div> <div>Interrupt Address</div> </div>	<div> <div>L7 L6 L5 L4 L3 L2 L1 L0</div> <div>Reset Interrupts</div> </div>
X7H	<div> <div>D7 D6 D5 D4 D3 D2 D1 D0</div> <div>Receiver Buffer</div> </div>	<div> <div>D7 D6 D5 D4 D3 D2 D1 D0</div> <div>Transmitter Buffer</div> </div>
X8H	<div> <div>D7 D6 D5 D4 D3 D2 D1 D0</div> <div>Port 1</div> </div>	<div> <div>D7 D6 D5 D4 D3 D2 D1 D0</div> <div>Port 1</div> </div>
X9H	<div> <div>D7 D6 D5 D4 D3 D2 D1 D0</div> <div>Port 2</div> </div>	<div> <div>D7 D6 D5 D4 D3 D2 D1 D0</div> <div>Port 2</div> </div>
XAH	<div> <div>D7 D6 D5 D4 D3 D2 D1 D0</div> <div>Timer 1</div> </div>	<div> <div>D7 D6 D5 D4 D3 D2 D1 D0</div> <div>Timer 1</div> </div>
XBH	<div> <div>D7 D6 D5 D4 D3 D2 D1 D0</div> <div>Timer 2</div> </div>	<div> <div>D7 D6 D5 D4 D3 D2 D1 D0</div> <div>Timer 2</div> </div>
XCH	<div> <div>D7 D6 D5 D4 D3 D2 D1 D0</div> <div>Timer 3</div> </div>	<div> <div>D7 D6 D5 D4 D3 D2 D1 D0</div> <div>Timer 3</div> </div>
XDH	<div> <div>D7 D6 D5 D4 D3 D2 D1 D0</div> <div>Timer 4</div> </div>	<div> <div>D7 D6 D5 D4 D3 D2 D1 D0</div> <div>Timer 4</div> </div>
XEH	<div> <div>D7 D6 D5 D4 D3 D2 D1 D0</div> <div>Timer 5</div> </div>	<div> <div>D7 D6 D5 D4 D3 D2 D1 D0</div> <div>Timer 5</div> </div>
XFH	<div> <div>INT REF TBE TRE BD PE OE FE</div> <div>Status</div> </div>	<div> <div>0 RS4 RS3 RS2 RS1 RS0 TME DSC</div> <div>Modification</div> </div>

Figure 3-9  
The internal MUART registers.

Here you see sixteen read registers and sixteen write registers. The read registers are accessed at their respective address during an MPU read operation ( $\overline{RD}$  low), while the write registers are accessed at their respective address during an MPU write operation ( $\overline{WR}$  low). For example, with the interface shown in Figure 3-8, an IN 5FH instruction will read the MUART Status Register and load its contents into the MPU accumulator, while an OUT 5FH instruction will store the MPU accumulator value to the MUART Modification Register. The most significant hex digit (5) in the instruction address is determined by the decoded address range for the MUART. The least significant hex address digit (F) accesses the MUART Status Register for the IN instruction, while it accesses the Modification Register for the OUT instruction. Of course, this least significant hex value can range from 0 to F to access any of the MUART registers. Thus, the MUART port address range will be from 50H through 5FH using the interface in Figure 3-8.

You will note from Figure 3-9 that some of the read/write register pairs at a given address are the same while some are different. Those that are the same have the same meaning for both the read and write operations. Those that are different have different meanings for a read versus a write operation. For instance, to access the Port 1 Data Register you use address X8H. A read operation at this address inputs the data present on the Port 1 input lines, while a write to address X8H outputs data on Port 1 output lines. Of course, the Port 1 I/O lines must be configured for input or output via the Port 1 Control Register at address X4H. On the other hand, when you read address XFH you get the MUART Status register, while writing to address XFH accesses the Modification Register.

You will also note from Figure 3-9 that the registers associated with parallel I/O Ports 1 and 2 have been tinted. These are the registers that have been discussed up to this point. The remaining MUART registers are used with the MUART interrupt, serial I/O, and timer functions. You will learn about these registers as we discuss these other functions of the MUART. As you will see, the "secret" to understanding the operation of any programmable I/O device, like the MUART, is found in the use of its internal registers.

## Initialization

Before you can use the MUART for data transfer, it must be initialized. Remember, the MUART is a programmable I/O device which means that it must be programmed for what it is to do. The process of programming a programmable I/O device is called **initialization**. Initialization is accomplished when the MPU executes an initialization routine which configures the internal registers of the device for a particular operation.



The MUART is initialized for parallel I/O by defining which port lines are to be used for input and which ones are to be used for output. Thus, the parallel I/O initialization routine must store values to the Port 1 Control Register to configure the Port 1 I/O lines and the Mode Register to configure the Port 2 I/O lines.

Here's a routine that will initialize the MUART by configuring Port 1 as an 8-bit input port and Port 2 as an 8-bit output port:

```
MVI A,00H
OUT 54H
MVI A,03H
OUT 53H
```

This routine assumes that the MUART has been decoded for port addresses 50H through 5FH per the interface circuit shown back in Figure 3-8. The routine begins by loading the accumulator with the value 00H then storing it out to port address 54H. From Figure 3-9 you see that the Port 1 Control Register is located at this address. Thus, logic 0s are stored to the Port 1 control register which configures all the port I/O lines for input. Next, the routine loads the accumulator with the value 03H and stores it to port address 53H. From Figure 3-9 you find the Mode Register located at this address and from Figure 3-3 you see that the lower two bits of this register must be set to configure all the Port 2 I/O lines for output. This translates to a hex value of 03H. That's all there is to it! The MUART is now initialized for 8-bit parallel input at Port 1 and 8-bit parallel output at Port 2.

After the MUART has been initialized, data is transferred between the MPU and the ports via the respective port Data Registers. For instance, suppose an input device is connected to Port 1 and supplies data to the system. To read this data, you simply execute an IN 58H instruction. This instruction will load the accumulator with the contents of port address 58H, which is the Port 1 Data Register (Ref. Figure 3-9). Likewise, suppose an output device is connected to Port 2 and you must transfer the contents of the accumulator to the output device. The instruction OUT 59H will do the job, because the Port 2 Data Register is located at port address 59H, right?

## Self-Test Review

1. What is the purpose of any programmable I/O device?
2. List the three primary functions of the 8256 MUART.
3. List the internal registers associated with the parallel I/O section of the MUART and briefly state their functions.
4. How must the MUART be connected to the 8085 for external interfacing?
5. How is a read register accessed at a given MUART address versus the write register at that same address?
6. When is I/O data latched by the MUART port lines?
7. To configure Port 1 of the MUART for output, the Port 1 Control Register must contain the hex value \_\_\_\_\_.
8. To configure lower half of Port 2 for input and the upper half for output, the Port 2 Mode Register must contain the hex value \_\_\_\_\_.
9. Assuming that the MUART is decoded for addresses 50H through 5FH using direct I/O, write a routine that will configure Ports 1 and 2 as specified in Questions 7 and 8.
10. Using the configuration provided by Question 9, what will the following instructions do?
  - A. OUT 58H
  - B. IN 58H
  - C. OUT 59H
  - D. IN 59H

## Answers

1. The purpose of any programmable I/O device is to simplify the problem of interfacing the MPU to external devices. In addition, the programmable I/O device relieves the MPU from performing routine I/O tasks and frees it up to perform more system related tasks.

2. Programmable parallel I/O, serial I/O, and timer functions.

3. Port 1 Control Register: Used to configure the Port 1 I/O lines as input or output.

Port 1 Data Register: Used to input and output data via the Port 1 I/O lines.

Mode Register: Lower three bits used to configure the Port 2 I/O lines for all input, all output, or a nibble combination of input or output.

Port 2 Data Register: Used to input and output data via the Port 2 I/O lines.

4. See Figure 3-8. The MUART address/data bus is connected directly to the 8085 multiplexed address/data bus. The *ALE*, *INTA*, *INT*, *RESET*, *CLK*, *WR*, and *RD* lines on the MUART connect directly to the corresponding lines on the 8085 MPU.

5. The read register is accessed at a given MUART address when the read (IN) operation is performed at that address, while the write register is accessed when a write (OUT) operation is performed at that address.

6. Only during output operations. Data coming into the MUART is not latched and must be read immediately by the MPU.

7. FFH.

8. 02H.

9. MVI A,FFH  
OUT 54H  
MVI A,02H  
OUT 53H

10.
  - A. Outputs the contents of the accumulator to Port 1.
  - B. Inputs the contents of the Port 1 Data Register to the accumulator. Note that since Port 1 is configured for output, this operation simply reads the output logic present on the port output lines.
  - C. Outputs the upper four bits of the accumulator to the upper four bits of Port 2 (*P24 - P27*). Does not affect the lower four bits of Port 2 because they are configured for input.
  - D. Inputs the lower four bits of Port 2 (*P20 - P24*) to the lower four bits of the accumulator. Inputs the output logic present of the upper four bits of Port 2 (*P24 - P27*) into the upper four bits of the accumulator, because *P24 - P27* are configured for output.

## USING A PROGRAMMABLE PARALLEL I/O PORT

Now that you are familiar with programmable parallel I/O ports in general and the MUART in specific, you are ready to examine some of the ways that these devices can be used. In this section, you will see how the MUART can be used to interface with displays and keyboards. You will start by examining how the MUART can be used to drive 7-segment displays.

### Driving 7-Segment Displays

Figure 3-10 shows how the MUART can be used to multiplex up to eight 7-segment displays. Assume that the MUART decoder is designed to respond to port addresses 50H through 5FH using direct I/O. Port 1 of the MUART is used to supply the 7-segment code to the displays. Port 2 is used to determine which display is selected. Here, discrete transistors are used to provide the required drive current for the displays.

The displays are common cathode types. To light the "a" segment of Display 1,  $Q_1$  must conduct through pin "a". Thus, a logic 1 must be applied to the base of  $Q_1$  and to pin "a" of Display 1.

Notice that both ports of the MUART must serve as outputs. Thus, during the initialization procedure, the Port 1 Control Register is set to FFH, and the Mode Register is set to 03H.

All displays are blanked by storing 00H in the Port 1 Data Register. This puts logic 0s on  $P10$  through  $P17$ . As a result, no segments of any display can light.

To display a specific eight-character message, the displays must be turned on one at a time in sequence. This is called *display multiplexing* and is controlled by Port 2 of the MUART. At the same time as the displays are being multiplexed by Port 2, the respective display codes must be stored to Port 1.

Here's how it works, Display 0 is first turned on by storing a 01H to Port 2 and the desired character code is stored to Port 1. Display 0 is then turned off and Display 1 turned on by storing a 02H to Port 2 then storing the desired character code to Port 1. Display 1 is then turned off and Display 2 turned on by storing a 04H to Port 2 then storing the desired character code to Port 1, and so on. This multiplexing process continues until the entire eight-character message has been displayed.

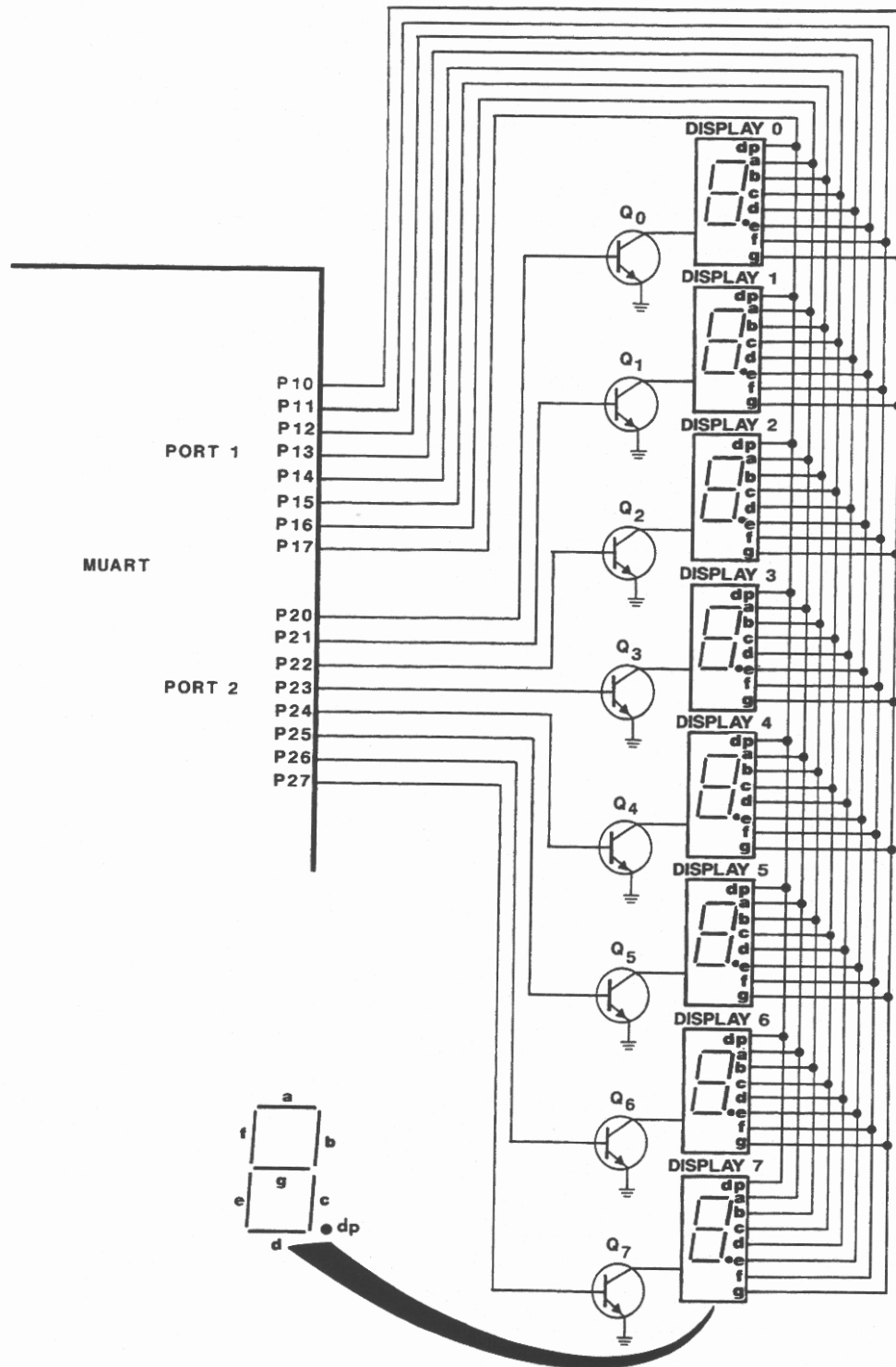


Figure 3-10  
Using the MUART to multiplex displays.

The display must be refreshed by repeating this multiplexing process at least 30 times per second. If not, the displays will appear as if they are flickering. By repeating the eight-character multiplexing operation at least 30 times per second, your retina retains the display image between each eight-character scan, resulting in a display that appears constant, without flicker.

Here's an algorithm that can be used for a display subroutine. We will assume that the eight 7-segment character codes to be displayed are stored in eight consecutive memory locations.

#### SUBROUTINE DISPLAY

BEGIN

Initialize the digit code to 01H.

Initialize display counter to 08H.

Load beginning display code address into index pointer register.

NEXT Load accumulator with digit code.

Output accumulator to Port 2

Load accumulator indirect via index pointer to get display code.

Output accumulator to Port 1.

Change digit code for next LED.

Increment the index pointer register.

Decrement the display counter.

Jump not zero to NEXT.

END

The algorithm begins by initializing three things: a digit code that will be used to turn on the LEDs in sequence, a display counter that will be used to control how many times the loop will execute, and an index pointer register that will be used to access the display codes stored in memory.

The digit code is initialized to 01H to turn on Display 0. This code must be changed to 02H, 04H, 08H, and so on, within the loop to turn on the remaining displays in sequence.

The display counter is initialized to 08H. This counter will be decremented down to 00H within the loop to control the number of times the NEXT loop is executed.

The index pointer register is loaded with the beginning address of the display code table in memory. This pointer will be used to load the accumulator with a given display code in memory. The pointer must be incremented each time through the loop to point to the next code in memory.

The loop begins by loading the accumulator with the digit code and storing it out to Port 2. This will turn on Display 0. The first character code is then loaded into the accumulator indirectly via the index pointer register. The code is then stored out to Port 1 to light Display 0 with the desired character. The digit code is then changed to 02H for Display 1, the index pointer is incremented to point to the next display code in memory, and the loop counter is decremented and tested for zero. Since at this point the loop counter is not zero, the NEXT loop repeats and Display 1 is lit with its desired character. The loop executes a total of eight times to light all eight LEDs in sequence.

Of course, SUBROUTINE DISPLAY must be called at least 30 times per second by the MPU to make the display message appear constant without flicker. How can the display message be changed? You're right, by changing the display codes stored in memory.

Now, here's an 8085 assembler routine that implements SUBROUTINE DISPLAY:

```

                PUSH B           ; Save BC on stack
                PUSH H           ; Save HL on stack
                PUSH PSW         ; Save A and CCR on stack
                MVI B,01H        ; Initialize digit code
                MVI C,08H        ; Initialize display counter
                LXI H,SGCODE      ; Initialize HL with beginning code address
NEXT   MOV A,B                 ; Move digit code to accumulator
        OUT 59H                ; Output digit code to Port 2
        RLC                    ; Rotate digit code left for next display
        MOV B,A                ; Move next digit code to B register
        MOV A,M                ; Move display code to accumulator
        OUT 58H                ; Output display code to Port 1
        INX H                  ; Increment display code pointer (HL)
        DCR C                  ; Decrement display counter
        JNZ NEXT               ; All displays done? If not, do next LED.
        POP PSW                ; Restore A and CCR
        POP H                  ; Restore HL
        POP B                  ; Restore BC
        RET                    ; Return to calling program

```

Here, you see that the registers to be used by the subroutine are first pushed onto the stack to save their existing contents. Register B is being used to hold the digit code, register C is used to hold the display count, and the HL register pair is being used as the index pointer register to point to the display codes in memory. We have assumed that the beginning address of the display codes is labeled SGCODE.



As you can see, the digit code is first sent out to Port 2 at address 59H to turn on the display. Recall that the Port 2 Data register in the MUART is located at address 59H, using the interface we developed earlier. The digit code value in the accumulator is then rotated left and moved to register B to prepare the code for the next display. This rotate left operation will produce the required digit code sequence of 02H, 04H, 08H, etc.

Once the LED is turned on by Port 2, the display code is loaded into the accumulator indirect via the HL pointer register. This code is then stored out to Port 1 at address 58H and the pointer register is incremented to point to the next display code.

The display counter in register C is decremented at the end of the loop and tested for zero. If not zero, the loop is executed again to illuminate the next display in the sequence. The display loop continues until all eight displays have been illuminated.

In order to give the illusion of a constant display, this subroutine must be called at least 30 times a second. The display must be refreshed by writing the message over and over again.

## Decoding Keyboards

Figure 3-11 illustrates how the MUART can be used to decode a 16-switch keyboard. The MUART is decoded to respond to addresses 50H through 5FH using direct I/O.

Notice that one switch is connected to each of the parallel I/O lines of the MUART. When a switch is open, the corresponding I/O line on the MUART is pulled up to a logic 1 by the pull-up resistor. When a switch is closed, the corresponding I/O line falls to a logic 0. In this application, both ports of the MUART must be initialized for input.

The problems associated with decoding the keyboard are the same as those discussed in Unit 1. Because this keyboard does not use interrupts, the MPU must scan the keyboard at regular intervals to catch a switch closure. Typically, it must read from addresses 58H and 59H several times each second.

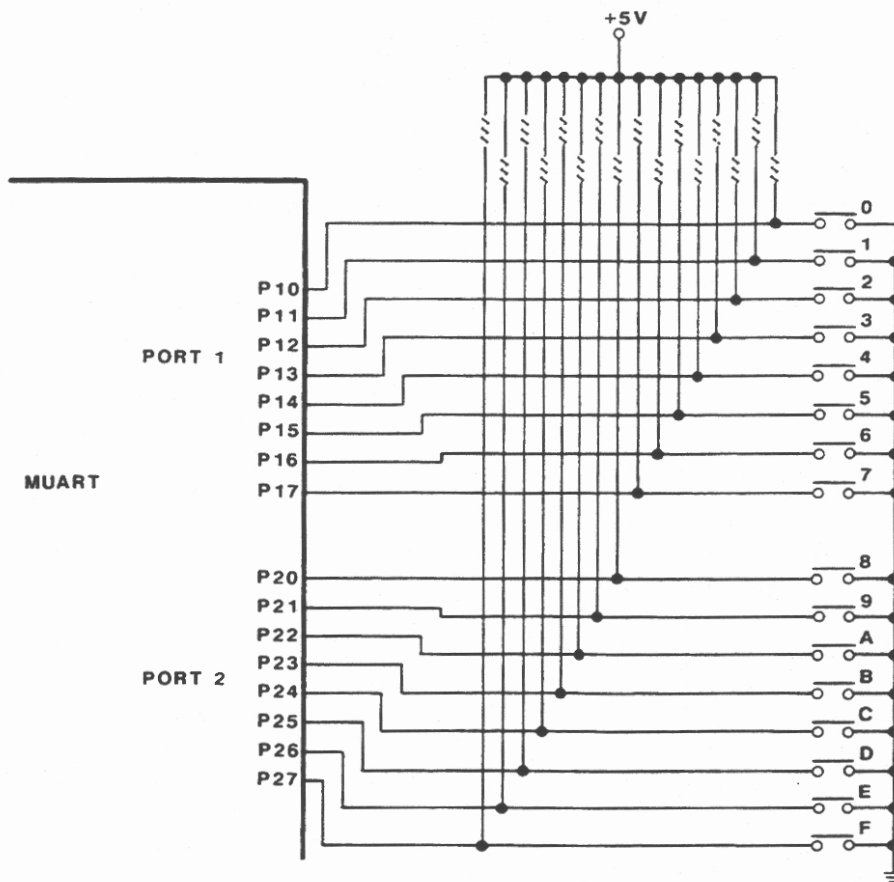


Figure 3-11  
Using the MUART to multiplex switches.

The MPU detects a switch closure by comparing the input data with FFH. If the input data is anything other than FFH, one or more of the switches is closed.

The MPU overcomes the switch bounce problem in the same way discussed in Unit 1. Also, the problems of rejecting multiple switch closures and producing an equivalent binary code can be accomplished using the same techniques discussed previously.

## DECODING A SWITCH MATRIX

The method shown in Figure 3-11 is a very straightforward technique of decoding switches. Using one MUART, this technique can only handle up to 16 switches. There are other techniques that use the MUART to greater advantage. An example is shown in Figure 3-12. Here again, the MUART is handling 16 switches. However, this time only one port of the MUART is used.

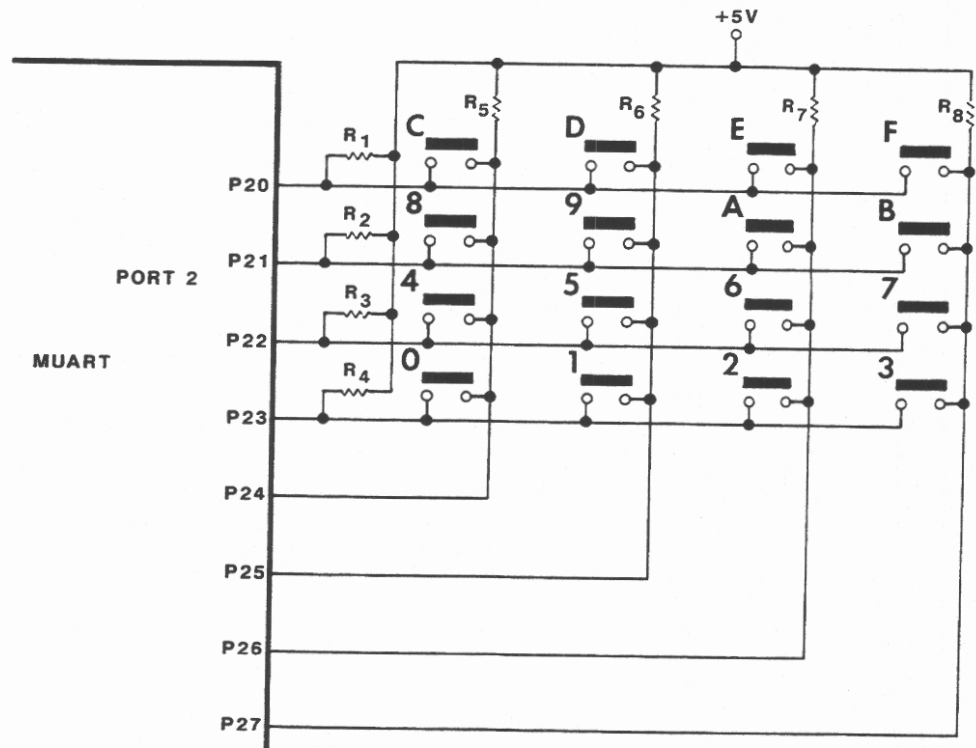


Figure 3-12

One port of the MUART can handle up to 16 switches using a switch matrix.

The 16 switches are arranged in a 4 by 4 matrix. Port 2 of the MUART is used to interface with the switches. Lines P20 through P23 are configured as input lines, while P24 through P27 are configured as output lines. This is an ideal application for Port 2, since storing a 02H to the Mode register configures the Port I/O lines in this manner (see Figure 3-3).

With no switches closed, P20 through P23 are pulled-up to a logic 1 by resistors R<sub>1</sub> through R<sub>4</sub>. Notice that P20 monitors switches C, D, E, and F. When all four switches are open, P20 is a logic 1. Even with one of the switches closed, P20 will still be at a logic 1 state if output lines P24 through P27 are at logic 1 states. And, these output lines (P24 through P27) are normally held at a logic 1 state.

Periodically, the MPU scans the keyboard to see if any switch has been closed. It does this by applying a logic 0 to one of the output lines and then checking for a logic 0 at one of the input lines.

A typical procedure might look like this. When the MUART is initialized, *P20* through *P23* are set up as inputs, while *P24* through *P27* are set up as outputs. The MPU scans the keyboard in this manner: *P24* is reset to 0 by storing EFH in the Port 2 Data Register. Next, Port 2 is read. If switch 0, 4, 8, or C is closed, its corresponding MUART input line will be low. For example, if switch 8 is closed, the 0 at *P24* will pull *P21* low. By examining lines *P20* through *P23*, the MPU can tell which switch (if any) is closed.

If no switch is closed in the first column, *P25* is reset to 0 by storing DFH in the Port 2 Data Register. The MPU can now check to see if switch 1, 5, 9, or D is closed.

The technique just described allows the MPU to handle a large number of switches with a single MUART. Using both ports, the MPU could handle an 8 by 8 matrix of 64 switches.

## Self-Test Review

11. Refer to Figure 3-10. Write a short routine that will initialize the MUART for the proper configuration.
12. Refer to Figure 3-10. Which port of the MUART determines which display is selected?
13. Refer to Figure 3-10. Assume that the MUART has been properly initialized. Write a simple routine that will display the numeral 4 on Display 7.
14. Refer to Figure 3-11. In order for this scheme to work, both ports of the MUART must be configured as \_\_\_\_\_.
15. Refer to Figure 3-11. How does the MPU tell that switch 9 is closed?
16. How can Port 2 of the MUART be configured as shown in Figure 3-12?
17. Refer to Figure 3-12. How does the MPU tell that switch 7 is closed?
18. If both ports of the MUART are used, what is the maximum number of switches that can be interfacing using the matrix technique?

## Answers

11. Assuming that the MUART is decoded for port addresses 50H through 5FH using direct I/O, a typical initialization routine would be:

```
MVI A,FFH
OUT 54H
MVI A,03H
OUT 53H
```

12. Port 2.
13. Keep in mind that  $Q_7$  must conduct and that pins b, c, g, and f must be high. Thus, the following code could be used:

```
MVI A,80H
OUT 59H
MVI A,33H
OUT 58H
```

14. Inputs.
15. The MPU reads in data from both ports of the MUART and compares this data with several different bit patterns. If switch 9 is closed, the bit pattern from Port 2 will be FDH.
16. The Port 2 I/O lines are configured in this manner during the initialization process by storing 02H in the Mode Register.
17. The MPU periodically resets line P27 to 0. If P22 is subsequently found to be 0, the MPU knows that switch 7 is closed.
18. Up to 64.

## I/O CONTROL TECHNIQUES

Communication between the MPU and an external device can be accomplished using one of two basic control techniques, *programmed control* or *interrupt control*. Under program control, all I/O transfers are controlled by the program. At scheduled intervals, the program reads data from input devices and writes data to output devices, regardless of whether or not the external device needs to be serviced. The obvious advantage here is simplicity and synchronization, since the I/O event is programmed into the system software. However, you sacrifice efficiency for simplicity since, many times, the MPU is being "tied-up" performing unnecessary read/write operations.

A more efficient form of program controlled I/O is *polling*. With this form of I/O control, the external device activates a *status flag*. The status flag is normally a bit location which will set when activated by the external device. The MPU must periodically read the status flag to determine if service is needed. If the flag has been activated, the data transfer will then take place. Again, some efficiency is sacrificed since the MPU must periodically break from what it is doing to check status flags. In addition, the external device must wait until the MPU gets around to checking its status flag before it can transfer any data. Thus, there is poor response time from the MPU, especially as more system tasks are assigned to the MPU.

The most efficient form of I/O control is with the use of interrupts. With this form of control, the MPU is interrupted from its main task by the external device. However, the interruption only occurs when the external device actually needs service from the MPU. Furthermore, the external device generally gets a relatively fast response time from the MPU.

Most programmable I/O devices, like the MUART, have built-in features to handle both programmed control and interrupt control of I/O operations. Let's take a look at how both are accomplished within the MUART. We will begin with interrupt control.

### Interrupt Control of I/O Operations

When time is critical to both the MPU and external device, you must use interrupt control. The MUART is capable of generating interrupts to the MPU via its *INT* line when any of its parallel I/O, serial I/O, or timer functions need to be serviced. In this section, we will discuss interrupt control of parallel I/O operations. Then in the next two units you will learn how to provide interrupt control of the serial I/O and timer operations.

Interrupts are *asynchronous* to the MPU program execution. This means that they are not synchronized to the program execution and, therefore, can occur at any time. In other words, an external device can generate an interrupt to the MPU, regardless of what the MPU is doing. To service the interrupt, the MPU must break away from its current task, execute the appropriate interrupt service routine, and return to where it left off when the interrupt occurred.

Furthermore, many external devices cannot operate at the speed of the MPU. Take a printer, for example. Assume that the MPU can only send one character to the printer at a time. It takes several milliseconds for the relatively slow printer to print a character but only a few microseconds for the MPU to send a character. Remember, a millisecond is a thousand times longer than a microsecond. Thus, the MPU is operating at least a thousand times faster than the printer.

So, the MPU sends a character to the printer and must wait several milliseconds to send the next character. Its like you doing something today, then having to wait several thousand days to do the same thing again. If this is the case, wouldn't it be nice if you could be doing something in the mean time and be notified when the task can be performed again? The same is true of the MPU and printer operation.

To be efficient, the MPU needs to send a character to the printer then go off and perform other chores while the character is being printed. When the printer is done printing the character, it must notify the MPU (via an interrupt) that it is ready to print another character. The MPU then breaks away from what it is doing and sends a second character. The MPU is then free to return to its other chores until another printer interrupt is received for another character to be printed. This interrupting process is repeated until the entire message is printed.

The process just described synchronizes the fast MPU with the relatively slow printer through the use of asynchronous interrupts. This process is commonly known as *handshaking*. Thus the term handshaking, as applied to computer I/O, refers to a technique used to synchronize communication between the MPU and I/O device. For a *complete handshake* to occur, two-way communication must take place between the MPU and I/O device. In general, three synchronization signals are used: one which asks "are you ready?", another which acknowledges "yes", and a final signal that indicates that the "operation is complete".

There are both input handshakes for input operations and output handshakes for output operations. The printer example discussed above is an application for an output handshake. On the other hand, a complete input handshake might go something like this:

1. The input device generates an interrupt to the MPU, signalling that it has data ready for the MPU. In effect, the input device is asking the MPU, "are you ready for my data?"



2. When the MPU accepts the interrupt it will send a signal to the input device which says "yes", thus acknowledging the receipt of the request.
3. When the read operation takes place, the MPU will signal the input device that "the operation is complete". Thus, the handshake is complete and can be repeated again for another data transfer.

Notice how the two-way communication between the MPU and input device synchronizes the data transfer.

## HANDSHAKING WITH THE MUART

The 8256 MUART is capable of generating all the signals required for both input and output handshaking. For parallel I/O, Port 2 is always used for the data transfer, while Port 1 is used to generate the handshake signals. Let's see how.

### Input Handshake

Figure 3-13 shows how the MUART is connected to provide input handshaking. Here, you see that the parallel input data is being transferred via Port 2, while the handshake signals are provided by Port 1. The two handshake signals are the *Strobe* ( $\overline{STB}$ ) input signal at P10 and the *Input Buffer Full* ( $\overline{IBF}$ ) output signal at P11. These will be discussed shortly.

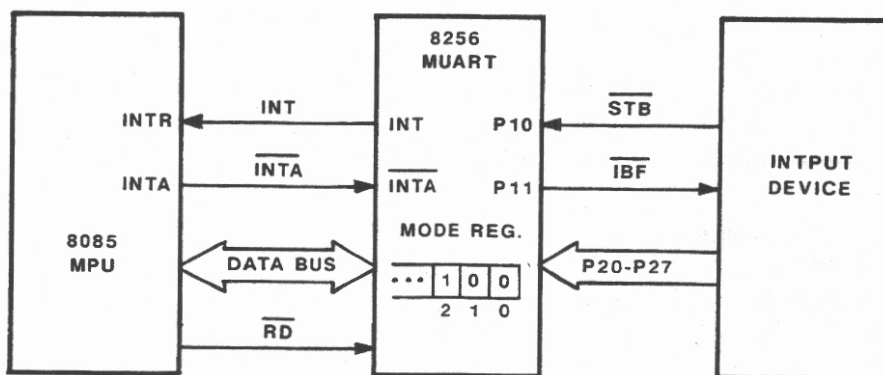


Figure 3-13  
An input handshake.

Notice on the MPU side of the MUART that there are three signal lines of interest in addition to the data bus. The  $\overline{INT}$  signal line from the MUART generates interrupts to the MPU and can be connected to any of the MPU interrupt input lines. The  $\overline{INTA}$  signal line from the MPU connects to the  $\overline{INTA}$  pin on the MUART and is used to provide an interrupt acknowledge. The  $\overline{RD}$  line from the MPU is connected to the  $\overline{RD}$  pin of the MUART and is used to signal the MUART when the MPU performs a read operation. Of course, other lines such as  $\overline{ALE}$ ,  $\overline{RESET}$ , etc. are also required for the MUART interface as discussed earlier, but the signal lines shown in Figure 3-13 are those of interest for the input handshake operation. Now, let's see how they all work together to provide an input handshake.

The timing diagram in Figure 3-14 shows it all. Let's follow through the timing of the handshake using the numbered references shown in the timing diagram.

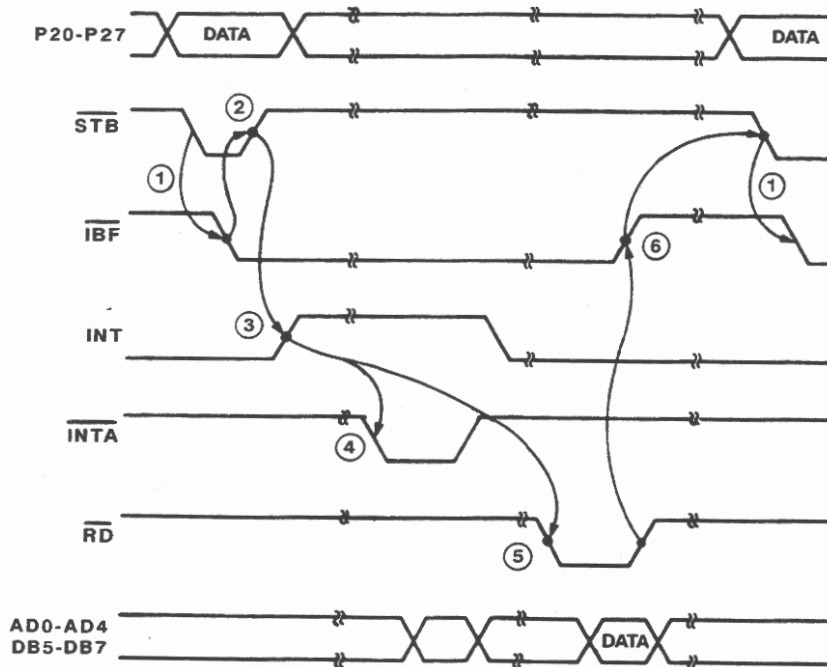


Figure 3-14

Timing of an input handshake.

- ① The input device pulls the  $\overline{STB}$  line low, indicating that it has data ready at Port 2. Almost immediately thereafter, the MUART acknowledges the request by activating  $\overline{IBF}$  (low).
- ② When the input device receives the  $\overline{IBF}$  acknowledge, it releases the  $\overline{STB}$  line.

- ③ The MUART interrupts the MPU via its *INT* line. This tells the MPU that input data is ready at Port 2 for transfer.
- ④ The MPU acknowledges the interrupt request by activating its  $\overline{INTA}$  interrupt acknowledge line.
- ⑤ The MPU reads Port 2 and activates the  $\overline{RD}$  line to indicate the read operation.
- ⑥ The MUART releases the  $\overline{IBF}$  line when the read operation occurs to tell the input device that its data has been read and the handshake is complete. The input device is then free to activate the  $\overline{STB}$  line again for the next byte of input data.

The above six step process is repeated for each byte to be transferred. When does it all stop? When the input device is done sending all its data. In other words, the handshake is initiated by the input device. When it is done with the data transfer, it simply stops strobing the  $\overline{STB}$  line.

## Output Handshake

Figure 3-15 shows the signal lines of interest for an output handshake.

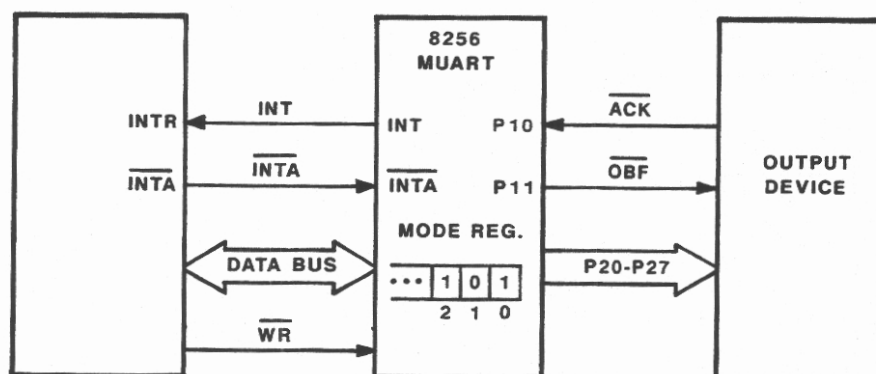


Figure 3-15  
An output handshake.

Again you see that the output data is transferred to the external device via Port 2. The two handshake signals are provided via the *P10* and *P11* lines of Port 1. These two signals are the *Acknowledge (ACK)* input signal at *P10* and the *Output Buffer Full (OBF)* output signal at *P11*.

On the MPU side of the MUART you see the *INT* (interrupt) and  $\overline{INTA}$  (interrupt acknowledge) signals that were also used for the input handshake. In addition, an output handshake requires the  $\overline{WR}$  (write) line from the MPU. Again, a timing diagram will show how everything works together to provide the handshake.

The timing diagram in Figure 3-16 illustrates the output handshake. Here's how it works:

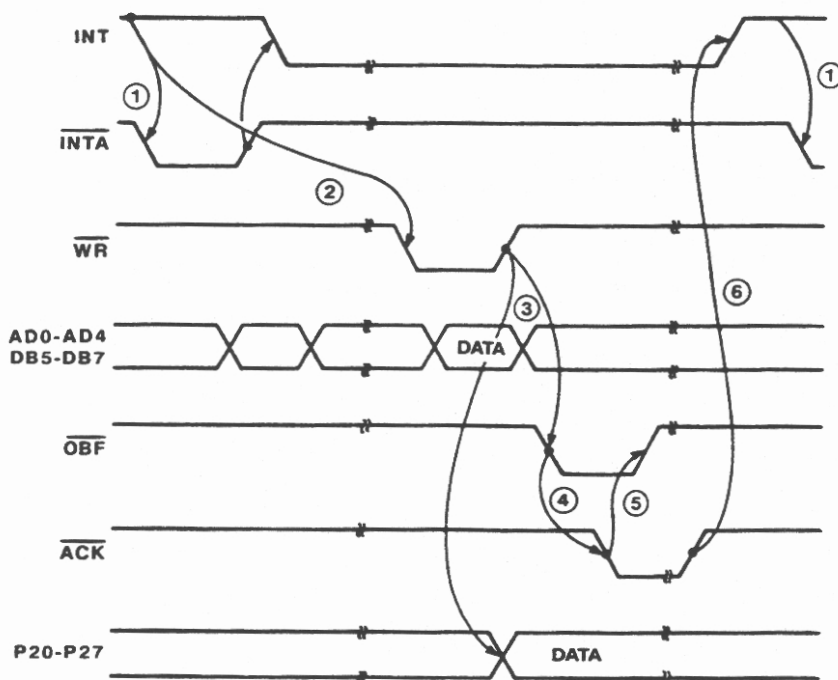


Figure 3-16  
Timing of an output handshake.

- ① The MUART interrupts the MPU via its *INT* line to indicate that the previous output data byte has been accepted by the external device and that the Port 2 output latch is empty. The MPU acknowledges the interrupt by activating its  $\overline{INTA}$  line.
- ② The MPU writes the next byte to Port 2 and activates its  $\overline{WR}$  line to indicate the write operation.
- ③ The write Port 2 operation causes the MUART to activate its  $\overline{OBF}$  line (low), indicating to the output device that the MUART output buffer is full and a new byte is available at Port 2.
- ④ The output device acknowledges the  $\overline{OBF}$  signal by pulling the  $\overline{ACK}$  line low.

- ⑤ When the MUART receives the acknowledge ( $\overline{ACK}$ ) signal, it releases  $\overline{OBF}$  back high.
- ⑥ When the output device actually takes the output data at Port 2 it acknowledges the data transfer by releasing the  $\overline{ACK}$  line back high. This causes another interrupt to be generated to the MPU via the  $INT$  line and the output handshake process is repeated for the next byte of data.

The above handshake process is started when the MPU writes data to Port 2 and repeated for each subsequent byte written to Port 2. The process is stopped when the MPU quits writing to Port 2.

### Configuring the MUART for Handshake Operations

The MUART is configured for handshaking by programming the Mode Register. Recall that the lower three bits of the Mode Register were used to configure the Port 2 I/O lines for all input, all output, or a nibble combination of input or output. In addition, these same lower three bits of the Mode Register are used to set-up the handshake operation. Table 3-1 shows the control logic for the lower three bits of the Mode Register again.

Table 3-1  
Using the Mode Register for Handshaking

P2C2	P2C1	P2C0	Mode	Direction	
				Upper	Lower
0	0	0	Nibble	Input	Input
0	0	1	Nibble	Input	Output
0	1	0	Nibble	Output	Input
0	1	1	Nibble	Output	Output
1	0	0	Byte Handshake	Input	
1	0	1	Byte Handshake	Output	
1	1	0	N/A	N/A	
1	1	1	Test		

To configure the MUART for input handshaking, you simply store a 1 0 0 in the lower three bits of the Mode Register. This automatically configures Port 2 for input, as well as defining  $P10$  as the  $\overline{STB}$  handshake line and  $P11$  as the  $\overline{IBF}$  handshake line. To configure the MUART for output handshaking, store a 1 0 1 in the lower three bits of the Mode Register. This automatically defines Port 2 as an output port and  $P10$  as the  $\overline{ACK}$  handshake line and  $P11$  as the  $\overline{OBF}$  handshake line. That's all there is to it! Once configured, the MUART controls the handshaking operation, freeing the MPU to perform more important system related tasks.

You might think that the configuration ends here, but there's more. You must define how the MUART will handle the interrupting process. In particular, you must know how the MUART works with the different MPU interrupts. This is the topic of our next discussion.

## INTERRUPT CONTROL USING THE MUART

You have just learned how the MUART provides parallel I/O handshaking through the use of interrupts. Parallel handshaking is just one interrupting source for the MUART. In addition to handshaking, the MUART also has other sources of interrupts due to its serial I/O and timer functions. In fact, there are 12 separate interrupt sources for the MUART as listed in Table 3-2.

Table 3-2  
MUART Interrupt Sources and Priority Levels

Interrupt Source	Priority Level	
Timer 1	Level 0	Highest
Timer 2 or P17	Level 1	
External (EXTINT)	Level 2	
Timer 3 or Timers 3 & 5	Level 3	
Serial Receive	Level 4	
Serial Transmit	Level 5	
Timer 4 or Timers 2 & 4	Level 6	
Timer 5 or Parallel Handshaking	Level 7	Lowest

From Table 3-2 you can distinguish 12 separate interrupt sources. Ten of these sources are from internally generated activities associated with parallel handshaking, serial I/O, or timer operations. Two of these sources are external to the MUART: the *P17* interrupt and the *External* interrupt sources. Up to this point, you are only familiar with the parallel handshaking interrupt source. The additional interrupt sources will be discussed as you learn about serial I/O and programmable timers. However, you will find that, although the source of the interrupt might be different, the way the MUART handles the interrupt is basically the same.

You will also note from Table 3-2 that the interrupt sources are prioritized into eight levels, from Level 0 with the highest priority to Level 7 with the lowest priority. The reason for the different levels and priorities will become apparent soon.

In Unit 1 you learned about the interrupt features of the 8085 MPU. You found that the 8085 has five hardware interrupts: *INTR*, *RST 7.5*, *RST 6.5*, *RST 5.5*, and *TRAP*. Recall that the *INTR* interrupt is a maskable interrupt that requires external logic to generate a hardwired RST (restart) instruction to locate the interrupt service routine. The *INTA* (interrupt acknowledge) signal from the MPU is used to enable the external logic to generate the RST instruction opcode onto the data bus. There are eight RST instruction possibilities that cause the MPU to go to eight unique call locations where a short service routine or JUMP instruction must be located.

On the other hand, the *RST 7.5*, *RST 6.5*, *RST 5.5*, and *TRAP* interrupts do not employ the *INTA* signal and external logic. These interrupts cause the MPU to automatically vector to specific memory locations where short service routines or JUMP instructions are located.

Remember that the MUART only has one interrupt output line (*INT*) that must be connected to one of the five 8085 interrupt input lines. It turns out that the MUART has the capabilities to handle both types of 8085 interrupts: *INTR* which uses the *INTA* interrupt acknowledge signal and the others which do not employ this signal. As a result, our discussion of interrupt control within the MUART will be divided into these two fundamental categories of 8085 interrupts.

### Using the 8085 *INTR* Interrupt Input

Remember that when you use the 8085 *INTR* interrupt input, you must use the *INTA* interrupt acknowledge signal to generate an RST opcode onto the data bus for interrupt vectoring. In Unit 1, you learned how to hardwire external logic to generate the required RST opcode. Now with the MUART, this external logic is not required, since the MUART is capable of generating the RST logic.

In Figures 3-13 and 3-15 we connected the MUART *INT* interrupt output line to the 8085 *INTR* interrupt input line for parallel handshaking. In addition, the MUART *INTA* line was connected to the 8085 *INTA* line. We said that when an interrupt occurred due to handshaking the MPU will vector to the appropriate interrupt service routine. The question now becomes: How?

Well, here's how it works. When an interrupt signal is generated on the MUART *INT* line, it is due to one of the 12 sources listed in Table 3-2. Of these 12 sources, only eight can be operational at any given time. These eight operational sources are designated as Level 0 through Level 7. For example, the Parallel Handshaking interrupt is designated as a Level 7 interrupt. When the MUART receives the *INTA* interrupt acknowledge signal from the MPU due to an *INTR* interrupt input to the MPU, it automatically places a RST opcode on the data bus. Now, there are eight RST opcodes and eight interrupt levels within the MUART. The particular RST opcode that is placed on the data bus depends on the interrupt level. A Level 0 interrupt (Timer 1) causes the RST0 opcode to be placed on the data bus, while a Level 7 interrupt (Timer 5 or Parallel Handshaking) causes the RST7 opcode to be placed on the data bus. Of course, a Level 7 interrupt will result from a parallel handshake if the Mode Register is configured for an input or output handshake.

Once the respective RST instruction is placed on the data bus, the MPU vectors to the memory address designated by the RST instruction. For instance, a Parallel Handshake interrupt places and RST7 instruction opcode on the data bus, which causes the MPU to vector to address 0038H. At this address you will typically place a JUMP instruction that causes the MPU to jump to a service routine to handle the parallel input or output operation. That's all there is to it! The MUART takes care of the RST opcode generation without any additional external logic.

Table 3-3 provides a listing of restart addresses versus MUART interrupt levels.

Table 3-3  
MUART Interrupt Levels and Their  
Corresponding Restart (RST) Addresses

Interrupt Level	Restart Address
Level 0	0000H
Level 1	0008H
Level 2	0010H
Level 3	0018H
Level 4	0020H
Level 5	0028H
Level 6	0030H
Level 7	0038H



## Using the *RST 7.5*, *RST 6.5*, *RST 5.5*, or *TRAP* Interrupt Inputs

Recall that these 8085 interrupts each have their own designated call locations where the MPU automatically vectors to obtain a JUMP instruction that locates the respective interrupt service routine. The respective call locations are listed again in Table 3-4 for your reference.

Table 3-4  
8085 Interrupts and Their Call Locations

Interrupt	Call Location
TRAP	0024H
RST 7.5	002CH
RST 6.5	0034H
RST 5.5	003CH

Interrupt handling by the MUART is different when you connect the MUART *INT* output line to any one of these MPU interrupt input lines. The reason is that the MPU *INTA* interrupt acknowledge signal is not employed with these interrupts. When this is the case, the *INTA* line on the MUART must be tied high. Now, the problem is this: How can the eight different MUART interrupt levels be vectored to separate interrupt service routines when only one interrupt call location is available for any given interrupt input line?

Here's how. The MUART contains an internal register called the **Interrupt Address Register** (Ref. Figure 3-9). The Interrupt Address Register is a read-only register located at port address X6H. Using our 8085/MUART interface back in Figure 3-8 the Interrupt Address Register would be located at port address 56H.

When an interrupt is generated due to any one of the eight different MUART interrupt levels, the MUART loads the Interrupt Address Register with four times the interrupt level. Thus, if a Level 7 Parallel Handshaking interrupt occurred, the MUART address register would contain the hex value  $4 \times 7 = 1CH$ .

Now, when the MPU receives a *RST 7.5*, *RST 6.5*, *RST 5.5*, or *TRAP* interrupt from the MUART, it will jump to respective service routine address as a result of a JUMP instruction located at the respective interrupt call location. One of the first instructions in the interrupt service routine must read the MUART Interrupt Address Register to obtain the interrupt level value. This value can then be used as an offset to jump to an interrupt vector table which contains JUMP instructions to the appropriate interrupt service routines.

For example, suppose we use the *RST 7.5* interrupt input on the MPU. When the MPU receives a *RST 7.5* interrupt from the MUART, it automatically vectors to address 002CH (Ref. Table 3-4). Now, at this point the MPU doesn't know which of the eight interrupt levels caused the interrupt. So, at address 002CH we insert a JUMP instruction that causes the MPU to jump to a *RST 7.5* service routine. The service routine will consist of the following instructions:

IN 56H	; Input interrupt level value
MOV L,A	; Move it to L register
XRA A	; Clear accumulator
MOV H,A	; Clear H register
LXI B,TABLE	; Load BC with begin table address
DAD B	; Add table address to level value
PCHL	; Transfer vector address to PC

The first service routine instruction reads the Interrupt Address Register of the MUART. We are assuming that the MUART is decoded for port addresses 50H-5FH. Recall that the Interrupt Address Register contains the interrupt level value. This value is then moved from the accumulator to the L register via the MOV L,A instruction. The accumulator is then cleared (XRA A) and moved to the H register. The HL register pair now contains the 16-bit representation of the interrupt level value. Next, the LXI B,TABLE instruction loads the BC register pair immediate with the beginning address of the interrupt vector table. We are using the label TABLE here, but your routine would have an actual two-byte address value. The DAD B instruction then adds the contents of the BC register pair to the HL register pair. In effect, this is adding the beginning vector table address to the interrupt level value to obtain a vector table address. The PCHL instruction then loads this vector table address into the program counter and causes the MPU to go to this address. At this address, you have a JUMP instruction to jump to the respective interrupt level service routine (Level 0 - Level 7).

For instance, suppose our vector table begins as address 8000H and a Parallel Handshake (Level 7) interrupt occurs. This creates an interrupt level value of  $4 \times 7$ , or 001CH, in the Interrupt Address Register. The above *RST 7.5* interrupt service routine would then add the beginning table value to the interrupt level value to obtain a vector table address of  $8000H + 001CH = 801CH$ . At address 801CH we would insert a *JUMP* instruction to jump to the Parallel Handshake service routine. The diagram in Figure 3-17 summarizes this vectoring process.

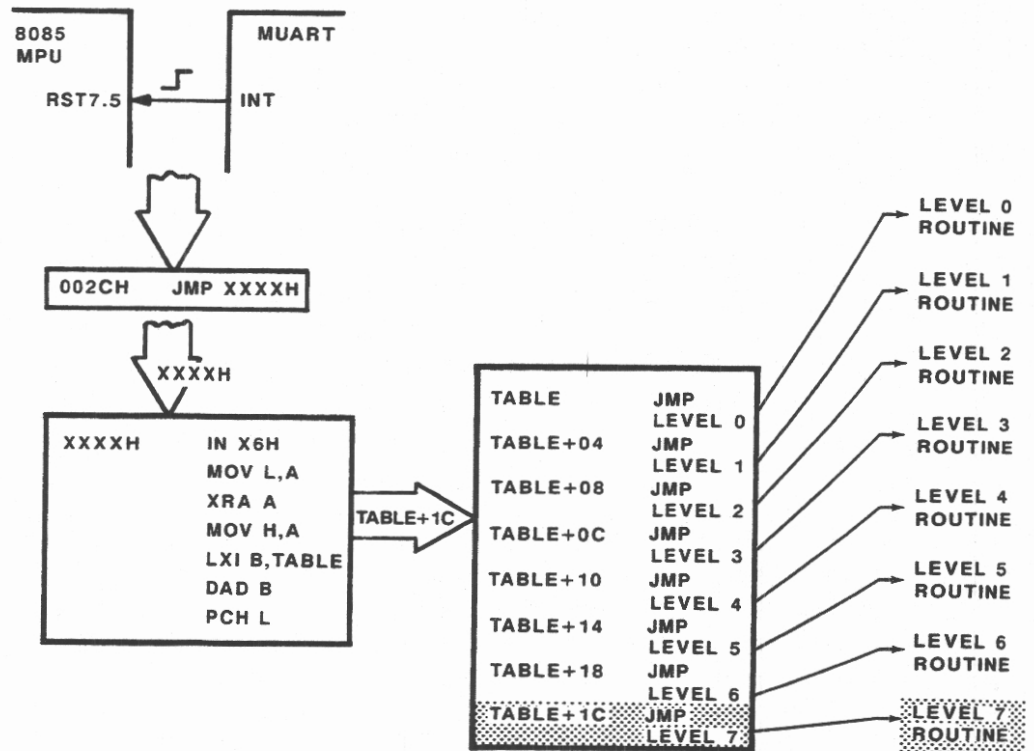


Figure 3-17

Interrupt vectoring using the *RST 7.5* interrupt input.

## CONFIGURING THE MUART FOR INTERRUPT SERVICING

Now, the last thing you need to know is how to configure the MUART to use the *INTR* mode of vectoring or the *RST 7.5, 6.5, 5.5, TRAP* mode. Recall that the *INTR* mode requires the use of the *INTA* interrupt acknowledge signal, while the others don't use this signal. If you look at the MUART register layout back in Figure 3-9, you will see an *IAE* (*Interrupt Acknowledge Enable*) bit in *Command Register 3* at port address X2H. This *IAE* bit is in the bit 5 position of Command Register 3 as shown in Figure 3-18.

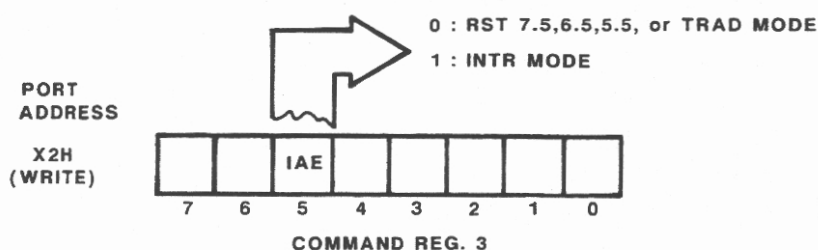


Figure 3-18

The *IAE* (Interrupt Acknowledge Enable) bit of Command Register 3 selects the interrupt mode for the MUART.

To use the *INTR* MPU interrupt mode, you must set the *IAE* bit (bit 5) in Command Register 3. To use the *RST 7.5, 6.5, 5.5, or TRAP* MPU interrupts, you must clear the *IAE* bit. The *IAE* bit is automatically cleared when the MUART is reset via its *RESET* line. So, if the MUART *RESET* line is connected to the system *RESET*, resetting the system will automatically place the MUART in the *RST 7.5, 6.5, 5.5, TRAP* interrupt handling mode. To obtain the *INTR* mode, you must set the *IAE* bit during the MUART initialization routine.

## INTERRUPT ENABLING WITHIN THE MUART

The last thing we need to discuss about interrupts in this section is the interrupt enabling feature within the MUART. You are now aware that the MUART is capable of handling eight separate interrupt levels that are associated with the parallel I/O, serial I/O, and timer operations of the MUART. There are registers within the MUART that allow you to enable and disable any of these interrupt levels. These are the *Set Interrupts Register* located at port address X5H and the *Reset Interrupts Register* located at port address X6H. Both are write-only registers.

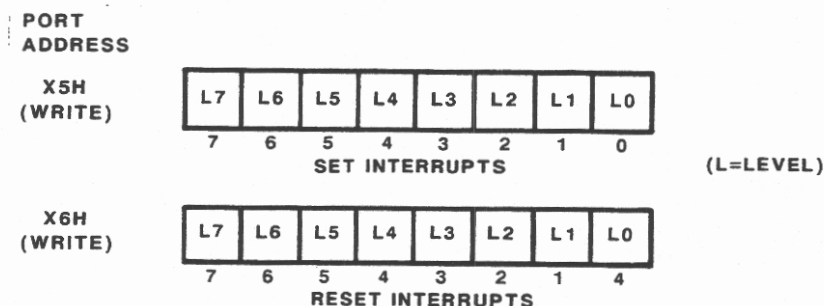


Figure 3-19

The Set and Reset registers are write-only registers within the MUART used to enable and disable the eight interrupt levels.

Look at Figure 3-19 and you will see that each bit within the registers is associated with a corresponding interrupt level. To enable a given interrupt level you must write a logic 1 to its bit within the Set Interrupts Register at port address X5H. To disable a given interrupt level you must write a logic 1 to its bit within the Reset Interrupts Register at port address X6H. Thus, to enable the Parallel Handshaking interrupt feature you must write a logic 1 to bit 7 in the Set Interrupts Register. To disable the interrupt feature for Parallel Handshaking you must write a logic 1 to the Reset Interrupts Register. Of course, disabling the interrupt feature for a given level interrupt would require the use of polling to handle that function.

Since the Set Interrupts and Reset Interrupts registers are write-only registers, another read-only register must be available to determine which level interrupts are enabled and which are disabled. This is the purpose of the *Interrupt Enable Register* at port address X5H. This is a read-only register that allows you to determine which interrupts are enabled. When this register is read, a logic 1 in a given bit position means that the corresponding interrupt level is enabled.

As a summary to this section, Figure 3-20 shows the MUART interrupt registers. You should now be familiar with the use of each register.

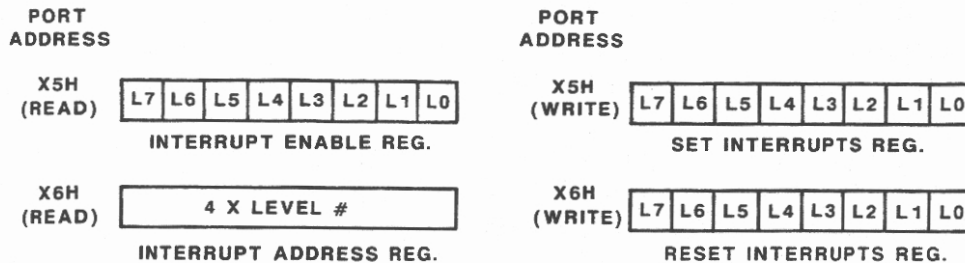


Figure 3-20  
The MUART interrupt registers.

## THE P17 AND EXTINT INTERRUPTS

Back in Table 3-2 you saw a *P17* interrupt source at Level 1 and an *EXTINT* source at Level 2. The *P17* interrupt is a low-to-high *edge triggered* interrupt source available on the *P17* line of Port 1. The *EXTINT* interrupt is a high level triggered interrupt source available as a separate pin on the MUART.

The *P17* line of Port 1 can be used as a separate interrupt input line when not used for other purposes. This is accomplished by setting the *BITI* bit (bit 2) in *Command Register 1* as shown in Figure 3-21. Recall that the Level 1 interrupt is shared by the Timer 2 interrupt source and the *P17* interrupt source. When the *BITI* bit in Command Register 1 is cleared, the Level 1 source is Timer 2, while the Level 1 source becomes *P17* when *BITI* is set. Regardless of which interrupting source is chosen, the Level 1 interrupt must be enabled via the Set Interrupts Register in order to use the interrupt feature.

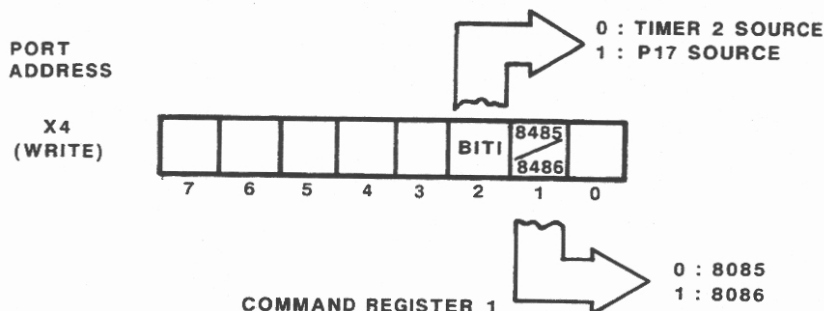


Figure 3-21  
The *BITI* bit in Control Register 1 determines the Level 1 interrupt source. The 8085/8086 bit selects between the 8085 or 8086 MPU modes.

In addition to the *BITI* bit in Figure 3-21, you will note that bit 1 in Command Register 1 is labeled 8085/8086. Clearing this bit puts the MUART in the 8085 interrupt mode, while setting this bit puts the MUART in the 8086 interrupt mode. For this course, we will always clear the 8085/8086 bit to place the MUART in the 8085 interrupt mode of operation. Furthermore, since resetting the MUART via its *RESET* line clears all its internal registers, the MUART is automatically placed in the 8085 mode of operation with a system reset.

Finally, Level 2 is permanently designated as the *EXTINT* source. The *EXTINT* interrupt is an external interrupt that is generated via the *EXTINT* pin on the MUART. The *EXTINT* interrupt is a *level triggered* interrupt. This means that the interrupting device must place a high level on the *EXTINT* pin for the Level 2 interrupt to be activated within the MUART. Of course, the Level 2 interrupt must be enabled via the Set Interrupts Register to enable this interrupt feature.

If a high level remains on *EXTINT* after the first servicing of the interrupt, the interrupt will be serviced again, and so on, until the high level is removed. There are several applications for level-sensitive interrupts, some of which will be demonstrated in the course Experiments. By the way, *EXTINT* is the only level sensitive interrupt source available on the MUART. All other interrupt sources are edge-triggered

## Polled Control of I/O Operations

If interrupts are not used, the only way to control the MUART operation is through polling. When using interrupts, an interrupt is generated via the MUART *INT* line whenever a given interrupt level needs service. However, with polling a status bit must take the place of the *INT* line. This status bit is the *INT* bit found in the bit 7 position of the Status Register. The *INT* bit in the Status Register sets (logic 1) whenever an enabled MUART feature needs service. Recall from Figure 3-9 that the Status Register is a read-only register located at port address XF8. By reading the Status Register and checking the status of bit 7, the MPU can determine if a MUART feature needs service.

So, the idea is for the MPU to periodically poll the *INT* bit in the MUART Status Register. When the *INT* bit is found set, a MUART feature needs service. However, the next problem is for the MPU to determine which feature needs service. This can be done by reading the Interrupt Address Register. Recall that the Interrupt Address Register contains a value that is four times the level needing service. Even though interrupts are not being employed when polling, the level value is still available in the Interrupt Address Register. Using this idea, we can write a simple polling algorithm like this:

#### SUBROUTINE POLL

```

BEGIN
    Read  MUART Status Register.
    Check INT bit status (bit 7).
    If INT set then
        Read MUART Interrupt Address Register.
        If level value = 00H then
            Jump to Level 0 routine.
        If level value = 04H then
            Jump to Level 1 routine.
        If level value = 08H then
            Jump to Level 2 routine.
        If level value = 0CH then
            Jump to Level 3 routine.
        If level value = 10H then
            Jump to Level 4 routine.
        If level value = 14H then
            Jump to Level 5 routine.
        If level value = 18H then
            Jump to Level 6 routine.
        If level value = 1CH then
            Jump to Level 7 routine.
    Else
        Return
END
    
```

To use this polling method, the various levels must have their interrupts enabled via the Set Interrupts Register, even though interrupts are not being employed.



There is another way to poll the MUART by polling each function separately. For example, when using the parallel handshaking function, you can poll the *P10* input line. Recall that the *P10* line of Port 1 is defined as an input handshake line for both input handshaking and output handshaking. For input handshaking, the *STB* signal is at *P10*, while the *ACK* signal is at *P10* for output handshaking. Thus, the polling subroutine can read Port 1 and check the *P10* status to see if a device needs to be serviced.

The serial I/O and timer functions of the MUART also can be polled separately. You will see how this can be done when we discuss these MUART functions in subsequent units.

## Self-Test Review

19. State the two basic control techniques used for data transfer within a microprocessor system.
20. Polling is a form of \_\_\_\_\_ control and always involves checking a \_\_\_\_\_.
21. What are two disadvantages of programmed I/O?
22. The most efficient form of I/O control is \_\_\_\_\_ control.
23. Why is handshaking required with many I/O devices?
24. How are the MUART parallel port lines configured for an input handshake?
25. What value must be stored in the Mode Register to configure the MUART for output handshaking?
26. With an output handshake, what is the handshake response to the  $\overline{OBF}$  (output buffer full) signal?
27. Explain how the MPU knows which MUART interrupting source generated an interrupt when the *INTR* line is used as the interrupt input line to the MPU.
28. Explain how the MPU knows which MUART interrupting source generated an interrupt when the *RST 6.5* line is used as the interrupt input line to the MPU.
29. How can the interrupt level value in the Interrupt Address Register be used to access the proper interrupt service routine?
30. Explain how to determine which MUART source needs service when using polling?

## Answers

19. Program control and interrupt control.
20. Program control and always involves checking a *status* flag.
21. Programmed I/O wastes MPU time and does not provide fast responses to I/O devices.
22. Interrupt
23. To synchronize the data transfer between the relatively fast MPU and slow I/O device.
24. Port 2 is configured for byte input, while the *P10* and *P11* lines of Port 1 are configured as handshaking signals. The *P10* line is used as input for the  $\overline{STB}$  handshake signal and the *P11* line is used as output for the  $\overline{IBF}$  handshake signal.
25. 05H
26. The  $\overline{ACK}$  (acknowledge) signal.
27. When the MUART receives the *INTA* signal from the MPU, it places an RST instruction opcode on the data bus. The RST opcode corresponds to the interrupt level that caused the interrupt. Thus, the Level 7 Parallel Handshake interrupt source causes the RST7 opcode to be placed on the data bus.
28. The MPU must read the MUART Interrupt Address Register. The value in the register will be four times the interrupt level. Thus, the Level 7 Parallel Handshake interrupt source results in an interrupt level value of  $4 \times 7$ , or 1CH, in the Interrupt Address Register.
29. The interrupt value is used as an offset and added to the beginning address of an interrupt vector table. The resulting sum points to the proper table address where a JUMP instruction is inserted to jump to the required interrupt service routine.
30. The MPU must periodically poll the *INT* bit (bit 7) in the Status Register to see if a MUART function needs service. If the *INT* bit is set, the MPU must read the Interrupt Address Register to determine which function needs the service.

## UNIT SUMMARY

The purpose of a programmable I/O device is to simplify the problem of interfacing the MPU to external peripheral devices and provide I/O flexibility through software configuration and control. Many of today's programmable devices, like the MUART, include parallel I/O, serial I/O, timer, and interrupt handling functions integrated onto a single IC.

The parallel I/O section of the MUART includes two 8-bit ports that can be configured for input or output. Port 1 is bit programmable via the Port 1 Control Register, while Port 2 is nibble programmable via the MUART Mode Register. An initialization routine must be executed to configure the port lines for input or output as desired. Once configured, writing to output port lines latches the data written to those lines. Reading input port lines gates the logic on those lines to the MPU data bus.

All programmable I/O devices employ internal registers that are used to control the operation of the device. The 8256 MUART includes 16 read-only registers and 16 write-only registers. The read-only registers are accessed via a read operation on the MUART, while the write-only registers are accessed via a write operation on the MUART. Some of the read/write register pairs at a given address are the same, while others take on a different meaning for a read versus a write operation.

The parallel I/O ports of the MUART can be used to multiplex up to eight 7-segment LED displays. Port 1 is used to supply the 7-segment code, and Port 2 is used to determine which display is selected. The two parallel I/O ports can also be used to decode a switch column of up to 16 switches or a switch matrix of up to 64 switches. In the latter case, one port is used as an output port to select a given switch column, while the other port is used as an input port to read the switch rows.

Communication between the MPU and an external peripheral device can be accomplished using one of two control techniques: programmed control or interrupt control. Programmed control can be as simple as blindly reading/writing I/O ports, or extended into polling where the MPU checks status flags to determine if service is needed. With interrupt control, the external device signals the MPU when it needs to be serviced. Interrupt control is by far the most efficient form of I/O control, since the MPU is not tied-up performing unnecessary software operations. In addition, interrupt control generally results in a better MPU response time to the external device.

Interrupt control of I/O operations using the MUART is accomplished by connecting the MUART *INT* line to one of the MPU interrupt input lines. In particular, connecting to the 8085 *INTR* interrupt input line requires the use of the 8085 *INTA* interrupt acknowledge signal to cause the MUART to place one of eight RST instruction opcodes onto the data bus so that the proper service routine can be accessed.

The MUART defines eight levels of interrupts that correspond to the eight RST instructions. The eight interrupt levels are derived from twelve interrupting sources, ten of which are internal to the MUART and two of which are external. The internal sources are derived from the parallel handshaking, serial I/O, and timer functions of the MUART. The two external sources are an edge-triggered interrupt input available at the *P17* port line, and a level-sensitive interrupt input provided by the *EXTINT* pin on the MUART.

When using the 8085 *RST 7.5*, *6.5*, *5.5*, or *TRAP* interrupt input lines, the MPU must use an interrupt level value obtained from the MUART Interrupt Address Register as an offset to access an interrupt vector table in memory. This process is required so that the MPU can locate the proper interrupt service routine in memory.

Both input and output handshaking can be provided via the MUART. Data is always transferred via Port 2, while *P10* acts as an input handshake line and *P11* acts as an output handshake line. A handshake synchronizes the transfer of data between the MPU and external peripheral device.

Finally, polled control of I/O operations can be provided via the MUART by using the *INT* bit in the Status Register as a status flag. If the *INT* bit is set, the MPU knows that service is needed, but must still read the Interrupt Address Register to determine which I/O function needs to be serviced.

## *Unit 4*

# **SERIAL DATA COMMUNICATIONS**

## CONTENTS

Introduction .....	4-5
Unit Objectives .....	4-6
Serial Communications .....	4-7
Serial Data Formats and Standards .....	4-7
Mark/Space .....	4-8
Frequency Modulation .....	4-9
Asynchronous/Synchronous Transmission and ASCII Coding .....	4-10
Serial Communications Channeling .....	4-13
Self-Test Review .....	4-15
Answers .....	4-16
Parallal/Serial Conversion .....	4-17
Software Conversion Via the 8085 .....	4-17
Serial Output .....	4-18
Serial Input .....	4-23
Serial I/O Timing .....	4-28
Hardware Conversion .....	4-30
Self-Test Review .....	4-33
Answers .....	4-35
Serial I/O Using the 8256 MUART .....	4-36
Functional Layout and Registers .....	4-36
Registers and Addressing .....	4-37
Transmit Buffer Register .....	4-38
Receive Buffer Register .....	4-39
Command Register 1 .....	4-39
Character Length (Bits L1, L0) .....	4-40
Stop Bit Length (Bits S1, S0) .....	4-40
Break-In Detect Enable (BRKI bit) .....	4-41
Command Register 2 .....	4-42
Baud Rate Select (Bits B3 - B0) .....	4-43
Internal Clock Prescaler (Bits C1, C0) .....	4-44
Even Parity (EP bit) .....	4-44
Parity Enable (PEN bit) .....	4-44
Command Register 3 .....	4-46
Reset (RST bit) .....	4-46
Transmit Break (TBRK bit) .....	4-47
Single Character Break (SBRK) .....	4-48
Receive Enable (RxE bit) .....	4-48
Bit Set/Reset (SET) .....	4-48
Modification Register .....	4-48
Disable Start Bit Check (DSC bit) .....	4-50
Transmission Mode Enable (TME bit) .....	4-50
Receiver Sample Time (Bits RS4 - RS0) .....	4-50

Status Register.....	4-51
Framing Error (FE bit).....	4-52
Overrun Error (OE bit) .....	4-52
Parity Error (PE) .....	4-52
Break/Break-In (BD) .....	4-53
Transmit Shift Register Empty (TRE).....	4-53
Transmit Buffer Register Empty (TBE).....	4-54
Receive Buffer Register Full (RBF) .....	4-54
Interrupt Pending (INT bit) .....	4-54
Interrupt Control of Serial I/O .....	4-55
Self-Test Review.....	4-59
Answers.....	4-60
Unit Summary.....	4-62





## ***Unit 4***

# **SERIAL DATA COMMUNICATIONS**

## **INTRODUCTION**

Up to this point, we have concentrated on parallel data communications because the microcomputer is basically a parallel device. However, many of the common peripheral devices that a microcomputer must communicate with are serial in nature. Such devices include CRT data terminals, printers, disks, and MODEMs. Thus, your microcomputer system must be capable of making parallel-to-serial and serial-to-parallel conversions when communicating with these devices.

In this unit, we will begin with a general discussion of serial communications. Then, you will learn how to provide parallel/serial conversions using both software and hardware techniques. First, you will learn about the internal serial I/O features of the 8085 MPU. Then, you will see how the 8256 MUART can function as a programmable serial communications interface to relieve the MPU of the serial conversion task.

## UNIT OBJECTIVES

1. Describe how serial data can be represented using both mark/space and frequency modulation techniques.
2. Explain the difference between asynchronous and synchronous serial data transmission.
3. Define what is meant by the term "baud rate" and how this term is applied to serial data communication.
4. List three different methods for channeling serial data.
5. Make parallel/serial conversions using software and the 8085 serial I/O lines.
6. Outline the steps necessary for software conversion of serial data to parallel data and parallel data to serial data.
7. Define the difference between a UART, USRT, and USART.
8. Describe the internal structure and register functions of the serial I/O section of the MUART.
9. Initialize and operate the MUART to perform parallel/serial data conversions.

## SERIAL COMMUNICATIONS

We live in a serial communications world. When you speak to someone, you first get their attention, then communicate your message, one word at a time. When you are through, you pause to indicate that you are done speaking. The same is true of reading and writing. You begin a sentence with a capital letter, then read or write one word at a time until you come to the period. These forms of human communication are serial, not parallel. Imagine how hard it would be to speak, read, or write in parallel. For example, try to speak a whole sentence at one time.

Human communication would be much more efficient if we could communicate in parallel since we could speak, read and write much faster. Computers, on the other hand, do communicate in parallel since it is much faster than serial communications. In fact, all the microcomputer data I/O you have studied up to this point in the course has been parallel data I/O.

So, why do we want our microcomputer to communicate in serial format? The answer comes from the fact that we are basically serial communication devices and the microcomputer is inherently a parallel communication device. Therefore, serial-to-parallel and parallel-to-serial translations must be made between the human/computer interface. As you will soon discover, most human/computer interfaces such as telephones, teletypewriters, CRT data terminals, printers, etc. are all serial devices. Another reason for serial communication in a computer system is economics. Serial I/O requires only two lines (transmit and receive) to communicate any number of data bits. However, with parallel I/O, a communication line must be provided for each data bit and control signals. Therefore, with an 8-bit microprocessor such as the 8085, at least eight parallel I/O lines are required. This requirement is obviously a disadvantage, especially when communicating over long distances.

### Serial Data Formats and Standards

There are two general methods used for binary serial data transmission in small microprocessor systems. They are: mark/space and frequency modulation (FM).

## MARK/SPACE

Mark/space computer data will represent a logic 1 state at one voltage or current level called a **mark**, and a logic zero state at another voltage or current level called a **space**. This mark/space format is shown in Figure 4-1. Mark/space format probably looks familiar to you because this is the way you have visualized digital data up to this point.

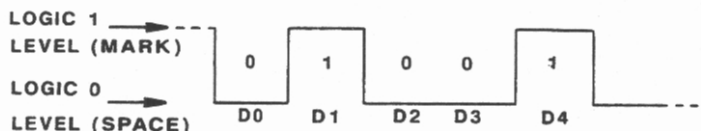


Figure 4-1

Mark/space serial data format.

There are several ways to represent a mark and a space with electrical signals. A few of the most common in microprocessor circuits are **TTL**, **CMOS**, **RS232C**, and **20 mA current loop**. The first three employ voltage levels, while the 20 mA current loop employs current levels.

With **TTL**, a logic 1 mark is represented by a +5V level, and a logic 0 space is represented by a 0V, or ground, level. **CMOS** represents a logic 1 with a voltage anywhere between +3V and +18V, with +12V being typical. A logic 0 is represented in **CMOS** by a 0V, or ground, level. Finally, the **RS232C** voltage standard, developed by the Electronics Industry Association (EIA), requires that a logic 1 be represented by a voltage anywhere between -3V to -25 V. A logic 0 is then represented by a voltage level anywhere from +3V to +25V. Many **RS232C** systems employ +12V and -12V to represent a logic 0 and 1, respectively.

The **20 mA current loop** uses current levels to represent logic. A logic 1 mark is represented by the presence of a 20 mA current level. A logic 0 space is represented by the absence of any current, or 0 mA.

Mark/space is usually employed for relatively short distances and low to medium data transmission rates. **TTL** and **CMOS** are applicable for distances up to 5 feet, **RS232C** up to 50 feet, and **20 mA current loop** up to one mile. In addition, any of these mark/space formats can be used for data transmission rates up to 9600 bits per second.

## FREQUENCY MODULATION

Frequency modulation is generally employed for longer transmission distances and higher speeds. In particular, frequency modulation is used when binary data must be transmitted via the telephone network. A device called a **MODEM** converts mark/space data to frequency modulated data for communication over the telephone network.

Frequency modulation requires that the two binary logic states be represented by two separate frequencies. Two common techniques used in the microprocessor industry are **frequency shift keying**, commonly called **FSK**, and **Biphase-M** modulation.

The FSK serial data format is shown in Figure 4-2. Here, a logic 0 is represented by a low frequency and a logic 1 represented by a higher frequency. Many MODEMs use a frequency of 1200 Hz to represent a logic 0 bit and a frequency of 2400 Hz to represent a logic 1 bit.

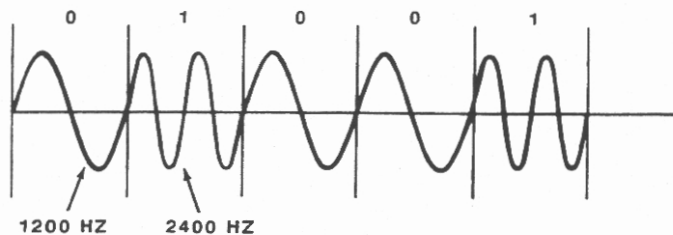


Figure 4-2  
Frequency shift keying, or FSK.

Biphase-M modulation is shown in Figure 4-3. This type of format combines level changes and frequency changes to represent binary data. Note that there is a level change at the beginning of each new bit. In addition, if the bit is a logic 1, there will be another level change at the center of the bit. Thus, the logic 1 frequency is twice that of the logic 0 frequency.

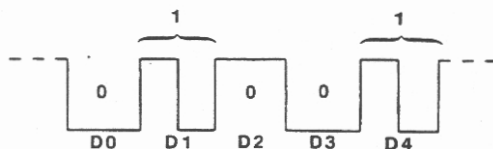


Figure 4-3  
Bi-phase serial data format.

Biphase-M modulation is often used for serial communication between microprocessors. This same format is also called FM, F/2F, and Manchester in different parts of the industry. In addition, it should be noted that this is a form of frequency modulation and not phase modulation. The term biphase is also sometimes used in connection with phase modulation. Phase modulation will not be discussed here, since it is not applicable to small microprocessor systems.

## Asynchronous/Synchronous Transmission and ASCII Coding

Once a serial standard is established, there are two common methods used for communicating the serial data between devices. They are, **asynchronous** and **synchronous** serial data communication. Asynchronous communication is by far the most common in small systems, and is used by all of the mark/space standards described previously.

The term **asynchronous** means "not clocked" and therefore there is no common clock between the transmitting and receiving devices. In a digital system, a clock is normally used to synchronize the transfer of data between different parts of the system. The clock defines the beginning and end of the data as well as the transmission speed. Without a common clock between the transmitter and receiver in a digital system, some other method must be used to synchronize the data transfer. Thus, with asynchronous serial communication, the data word is framed to define the beginning and end of the word in the same way that you frame a sentence with a capital letter and a period. Asynchronous communication is shown in Figure 4-4 using mark/space format. The beginning of the data word is defined by a **start bit**. The start bit is represented by a transition to a logic 0 level, or space, for one bit-time period. After the start bit, the eight data bits are transmitted, one bit at a time, beginning with the least significant data bit (D0). The end of the data word is then defined by a **stop bit**. The stop bit is represented by a logic 1 level, or mark, and may last for 1, 1 1/2, or 2 bit-time periods. This is normally a user-selectable option on both the transmitter and receiver. When no data is being transmitted, the serial line is held high (mark) until the next start bit is generated.

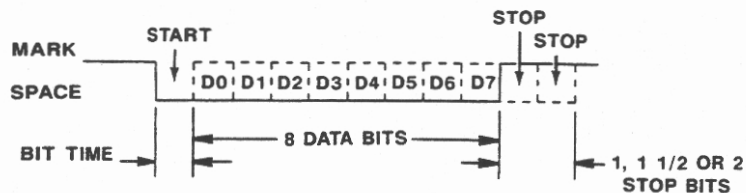


Figure 4-4

Asynchronous serial communication data format.

By framing the data word with start and stop bits, the receiver can detect the beginning and end of the word and thus establish synchronization with the transmitter without a **common** clock. This is not to say that the transmitter and receiver are not clocked. In fact, they are clocked, and must be clocked at the same frequency. The individual transmitter and receiver clocks establish the duration of each bit, or bit-time period, and therefore control the transmission speed.

Two terms are associated with serial transmission speed, **baud rate** and **data bits per second (DBPS)**. Baud rate is the number of bits being transmitted per second. However, with asynchronous transmission, this is not the number of data bits being transmitted per second since the eight data bits are framed by start and stop bits. Given a particular baud rate you can determine the bit-time period by simply taking its reciprocal. For example, a 9600 baud rate means that each bit lasts  $1/9600$  seconds or approximately 104 microseconds. Both the transmitter and receiver must be set to the same baud rate for proper serial communication to take place.

With asynchronous transmission, most of the data characters are sent in a standard code called **ASCII**, which stands for **American Standard Code for Information Interchange**. This code uses seven data bits to generate the numbers 0 through 9 and the upper and lowercase alphabet letters. In addition, many punctuation and mathematical symbols as well as nonprintable control characters can be generated. The ASCII code is shown in Figure 4-5.

The ASCII code reserves the most significant data bit (D7) for **parity**. Many times, a bit-error can occur in the serial transmission, especially over long distances. The parity bit is automatically set or cleared by the transmitter to make the data word contain an even or odd number of ones, depending on the system requirements. The receiver then checks the incoming data word for proper parity (even or odd). An even parity system requires that all the transmitted data words contain an even number of ones. An odd parity system requires an odd number of ones in the data word. Most serial transmitting and receiving devices can be optionally set for even, odd, or no parity. However, both the transmitter and receiver must be set for the same parity. It should be noted that a parity system will detect 1-bit errors, but 2-bit errors will go undetected, since a 2-bit error will result in the same parity word.



COLUMN		0	1	2	3	4	5	6	7
ROW	BITS 654 3210	000	001	010	011	100	101	110	111
0	0000	NUL	DLE	SP	0	@	P	\	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	\$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
10	1010	LF	SUB	*	:	J	Z	j	z
11	1011	VT	ESC	+	;	K	[	k	{
12	1100	FF	FS	.	<	L	\	l	!
13	1101	CR	GS	-	=	M		m	}
14	1110	SO	RS		>	N	^	n	~
15	1111	SI	US	/	?	O	_	o	DEL

Figure 4-5  
7-Bit ASCII Code.

The mark/space serial data word in Figure 4-6 shows what the asynchronous data stream would look like for the ASCII character "E" in an even-parity system. From Figure 4-5, you find that the 7-bit ASCII code for the letter "E" is 1000101. To make this an even-parity word, a logic 1 must be added in the most significant parity bit position. Therefore, the word becomes 11000101. Now, the serial system transmits the least significant bit first. Thus, the above bit pattern is reversed in Figure 4-6.

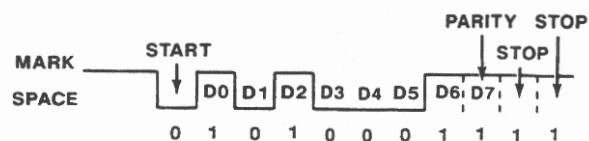


Figure 4-6  
Asynchronous data stream for ASCII character "E"  
with even parity and two stop bits.

As stated earlier, the other method of serial communication is **synchronous** communication. The term synchronous means "clocked" and, therefore, the transmitter and receiver are synchronized by a common clock source. No start or stop bits are needed to frame the data with this type of transmission. Instead, the transmitter and receiver are synchronized by an all 1's, or mark, preamble for a fixed number of clock cycles prior to the serial transmission. The preamble tells the receiver circuit when to expect the first bit of serial data. The serial data message is then sent in a continuous bit pattern, one bit per clock cycle, until the message is complete. When no data is being transmitted, special synchronizing (sync) characters are sent to maintain synchronization. In addition, the transmitter and receiver will normally use a handshake procedure to control the send/receive transmission process. The synchronous transmission technique is illustrated in Figure 4-7.

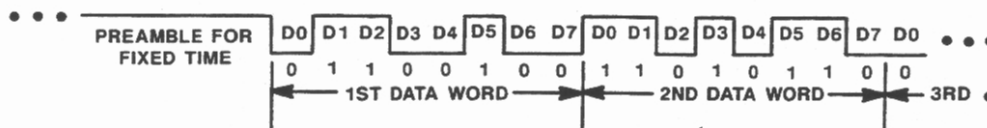


Figure 4-7

Synchronous serial data transmission.

Synchronous transmission is approximately 20% faster than asynchronous transmission since only eight bit-times are needed to transmit an 8-bit character. Asynchronous transmission would require ten or eleven bit-times for the same character. Therefore, synchronous transmission is used where very high speed transmission is required as in communication between high speed minicomputers and main-frame computers. Microcomputers are relatively slow-to-medium speed and use asynchronous communication almost exclusively.

## Serial Communications Channeling

You can channel serial I/O using one of three techniques: **simplex**, **half duplex**, or **full duplex**. Again, our point of reference is always the MPU when discussing data communication. Thus, data will travel from the MPU to a peripheral device on a serial transmit, or  $T_x$ , line. Conversely, data will travel from the peripheral device to the MPU on a serial receive, or  $R_x$ , line.

**Simplex** serial communication uses only one-way, one-line communication. The communication channel is either a transmit line ( $T_x$ ) or a receive line ( $R_x$ ), not both. **Half-duplex** serial communication uses one channel, like simplex, but two-way communication is provided on the single channel. Thus, with half-duplex, the single channel is a transmit line ( $T_x$ ) during an MPU transmit operation and a receive line ( $R_x$ ) during an MPU receive operation. **Full-duplex** serial communication uses two channels, with one channel providing the transmit line ( $T_x$ ) and the other providing the receive line ( $R_x$ ). The advantage of full-duplex over half-duplex is that with full-duplex, the transmit and receive operations can occur simultaneously. Simultaneous transmit and receive operations are not possible with half-duplex since the two operations must share a common channel. The three channeling methods are summarized in Figure 4-8.

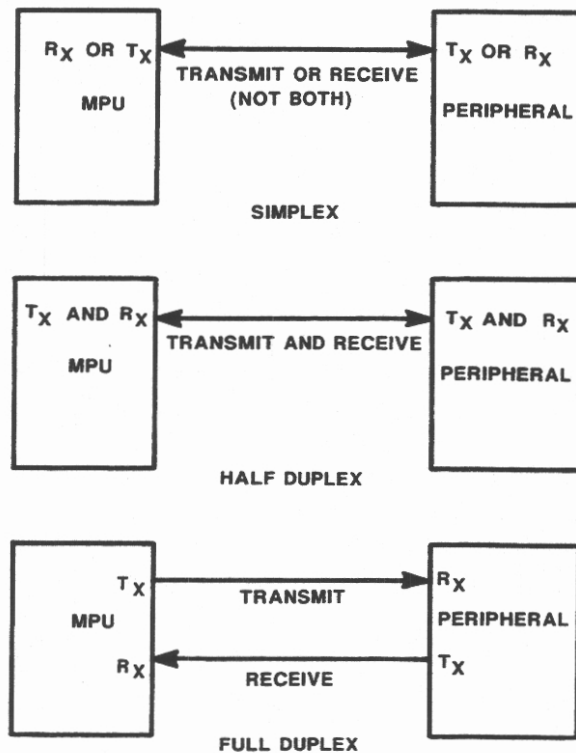


Figure 4-8  
Serial I/O channeling.

Most serial I/O peripherals are capable of either half-duplex or full-duplex operation. (You might note that simplex is just a special case of half-duplex.) These options are usually switch-selectable on the peripheral. Of course, both the MPU and peripheral must be configured for the same type of channeling.

## Self-Test Review

1. The two general modulation techniques used for binary serial data transmission are \_\_\_\_\_ and \_\_\_\_\_.
2. Another name for frequency modulation is \_\_\_\_\_.
3. List at least three standards that use mark/space serial format: \_\_\_\_\_  
\_\_\_\_\_
4. How are logic levels represented with the RS232C standard? \_\_\_\_\_  
\_\_\_\_\_
5. Two standards which use frequency modulation for serial data transmission are \_\_\_\_\_ and \_\_\_\_\_.
6. What are the two common methods used for communicating serial data between devices? \_\_\_\_\_  
\_\_\_\_\_
7. How does asynchronous transmission differ from synchronous transmission? \_\_\_\_\_  
\_\_\_\_\_
8. The most common method used for communicating serial data in microcomputer systems is \_\_\_\_\_.
9. A serial transmission speed of 300 baud means that the bit-time period is \_\_\_\_\_ seconds.
10. Baud rate and DBPS are always the same. (True or False?)
11. What is ASCII and where is it used? \_\_\_\_\_  
\_\_\_\_\_
12. How much faster is synchronous transmission over asynchronous transmission? \_\_\_\_\_  
\_\_\_\_\_
13. Three common ways used to channel serial communications are \_\_\_\_\_, \_\_\_\_\_ and \_\_\_\_\_.

## Answers

1. Mark-space and frequency modulation (FM).
2. Frequency-shift keying (FSK).
3. Three standards that use mark/space serial format are: TTL, RS232C, and 20 mA current loop.
4. The RS232C standard requires that a logic 1 be represented by a voltage level anywhere from -3V to -25V. A logic 0 is represented by a voltage level anywhere from +3V to +25V.
5. Bi-phase modulation and frequency shift keying (FSK).
6. The two common serial communication methods are asynchronous and synchronous data transmission.
7. With asynchronous transmission, the data word is framed by a start bit and one or more stop bits.
8. Asynchronous communication.
9. 3.33 milliseconds.
10. False, baud rate is the total number of bits transmitted per second and DBPS is only the number of data bits transmitted per second.
11. ASCII stands for American Standard Code for Information Interchange. It is an 8-bit (seven data bits + one parity bit) code used to represent the numbers 0 through 9 and the upper and lowercase alphabet letters. ASCII is the most common communication code used in this country.
12. Synchronous transmission is approximately 20% faster than asynchronous transmission.
13. Simplex, half-duplex, and full-duplex.

## PARALLEL/SERIAL CONVERSION

Because the microprocessor is inherently a parallel device, conversions between parallel and serial data format must be made when communicating with a serial peripheral device. As mentioned earlier, many standard peripheral devices such as CRT data terminals, printers, MODEMs, etc. require serial data. Thus, a parallel-to-serial conversion must be made when data is transmitted to one of these devices and a serial-to-parallel conversion must be made when data is received from such a device.

Parallel/serial conversions can be accomplished using both software or hardware techniques. A discussion of each follows.

### Software Conversion Via the 8085

The 8085 MPU has the unique ability to transmit and receive serial data via its *SOD* (Serial Output Data) and *SID* (Serial Input Data) lines. These two lines eliminate the need to construct special serial I/O ports external to the MPU. In fact, you can think of the *SOD* line as a 1-bit output port line and the *SID* line as a 1-bit input port line.

Imagine the *SOD* and *SID* lines as being connected to bit 7 (D7) of the 8085 accumulator as illustrated in Figure 4-9. That is, any serial data being transmitted by the MPU on the *SOD* line must be shifted through bit 7 of the accumulator. Likewise, any serial data being received by the MPU on the *SID* line must be input via bit 7 of the accumulator. Of course, there is no conflict since the serial transmit and receive operations do not take place at the same time.

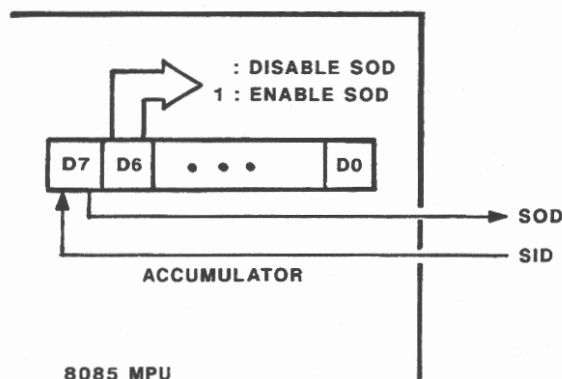


Figure 4-9

Serial I/O lines *SID* and *SOD* operate through bit 7 of the accumulator.

Notice also from Figure 4-9 that bit 6 (D6) of the accumulator is used to enable/disable the *SOD* output line. In other words, bit 6 of the accumulator must be set to transmit data via the 8085 *SOD* line. There is no such enabling/disabling feature for the *SID* input line.

The two instructions associated with the 8085 serial transmit and receive operations are *SIM* and *RIM*. You might recall that these instructions are also used for interrupt processing within the 8085 as discussed in Unit 1. *SIM* is used to set and clear the 8085 interrupt mask bits in the accumulator, while *RIM* is used to read the status of the interrupts. When used for serial I/O, *SIM* and *RIM* take on completely different meanings. For serial I/O, you can think of *SIM* as a 1-bit *OUT* instruction and *RIM* as a 1-bit *IN* instruction. Thus, executing a *SIM* instruction outputs the logic in the bit 7 position of the accumulator to the *SOD* serial output line. Of course, bit 6 of accumulator must be set to enable the *SOD* output line. On the other hand, executing a *RIM* instruction gates the logic on the *SID* serial input line into bit 7 of the accumulator.

Now, let's take a look at the software required to implement serial I/O via the 8085 *SOD* and *SID* lines. We will begin with serial output via the *SOD* line.

## SERIAL OUTPUT

Let's first construct a general algorithm to perform the serial output task. Then, we will refine the algorithm to be applicable to the 8085 MPU. We will assume that our serial output character consists of eleven bits as follows: a *START* bit, 7 ASCII character bits, a parity bit, and two *STOP* bits. Here's a general subroutine algorithm that will generate the serial output:

```
SUBROUTINE SERIAL OUT
  BEGIN
    Initialize counter to count the bits.
    Transmit START bit.
    Delay 1 bit time.
    Get the ASCII character to be transmitted.
  NEXT   Transmit character bit.
         Delay 1 bit time.
         Shift next bit into position.
         Decrement counter.
         If count not zero then
           Jump to NEXT.

         Transmit two STOP bits.

  END.
```

From this algorithm, it's quite clear what is happening. A counter is initialized as a loop counter to count the bit transmissions. The START bit is transmitted first, followed by a delay of 1 bit time, then the loop is entered to transmit each of the eight character bits. Within the loop, a given character bit is transmitted, a delay of one bit time is executed, the next bit to be transmitted is shifted into position, the counter is decremented and tested for zero. Once all eight character bits have been transmitted, the loop is broken and the two STOP bits are transmitted.

It is important to note that one of the main operations within the NEXT loop is a *shift* operation. In fact, all parallel/serial conversions involve shifting operations. For software conversions, shifting is accomplished using software shift instructions. For hardware conversions, shifting is accomplished using hardware shift registers. Look for the shifting operations in the material that follows. You will find that these are the key operations employed in parallel/serial conversions.

Another observation to be made in the above algorithm is the use of a delay between each of the bits to be transmitted. This delay establishes the bit transmission rate, which in turn determines the serial baud rate. Thus, the baud rate is altered by changing the amount of time delay between successive bit transmissions.

Now that you are familiar with the general transmission process, let's refine our algorithm to make it more applicable to microprocessor software in general, and the 8085 in particular. This general algorithm will need to be altered slightly to make it compatible with the 8085 architecture.

#### SUBROUTINE SERIAL OUT

##### BEGIN

Initialize a counter register to count the eleven bits.

Reset the Carry bit to 0.

##### NEXT

Load accumulator with value 80H.

Shift Carry into bit 7 and bit 7 into bit 6 of accumulator.

Transmit bit 7 of accumulator via the *SOD* line.

Delay one bit time.

Set the Carry bit to 1.

Load the ASCII character into the accumulator.

Shift bit 0 into Carry.

Save the accumulator contents in another register.

Decrement the bit counter.

Jump not zero to NEXT.

##### END.

Again, the key to the whole process is found in the shifting operations. Here, we are shifting the data to be transmitted through the MPU Carry bit. Let's desk-check the algorithm to see how it works. Look at Figure 4-10. The following discussion is keyed to this figure.



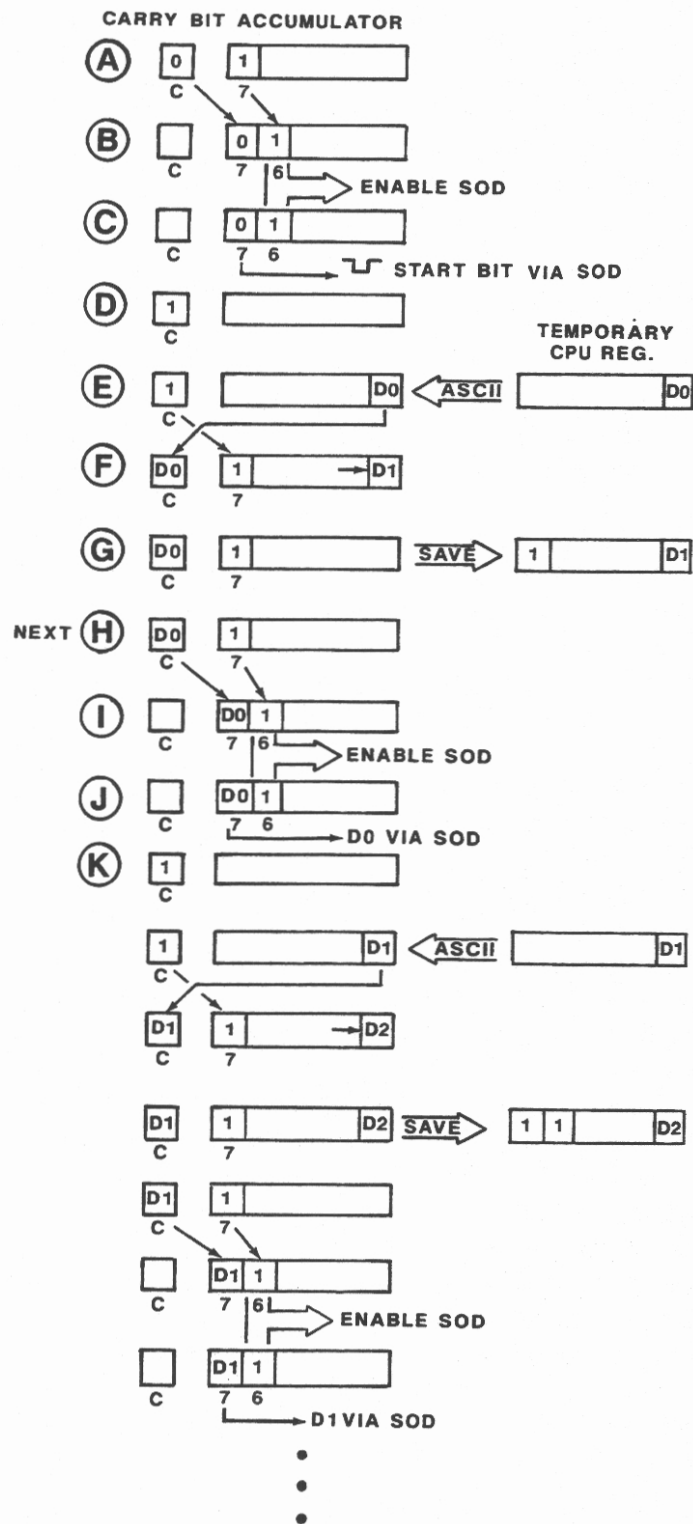


Figure 4-10  
Execution of the SERIAL OUT algorithm.

- Ⓐ Here you see that the Carry bit is cleared and the accumulator contains the value 80H, as required by the second and third instructions in the algorithm.
- Ⓑ Next, the figure shows the Carry bit shifted into bit 7 of the accumulator, and bit 7 is shifted into the bit 6 of the accumulator. This shifting operation does two things: Places a 0 in bit 7 of the accumulator in preparation for transmission as the START bit, and places a 1 in the bit 6 position to enable the *SOD* serial output line.
- Ⓒ The algorithm then outputs bit 7 to generate the START bit. A delay of one bit time is executed after the bit is transmitted.
- Ⓓ The Carry bit is set.
- Ⓔ The ASCII value to be transmitted is loaded into the accumulator.
- Ⓕ The accumulator is shifted right. This shifts bit 0 of the accumulator into the Carry bit. Now, the next bit to be transmitted (D0) is in the Carry bit.
- Ⓖ The current value of the accumulator is saved. At this point in the execution, the accumulator contains the shifted binary value. The algorithm then decrements the bit counter and jumps back to NEXT.
- Ⓗ At NEXT, the value 80H, or 1000 0000 binary, is loaded into the accumulator.
- Ⓘ The accumulator is then shifted again to bring the Carry bit into the bit 7 position of the accumulator. Bit 7 is shifted to the bit 6 position. Now, recall that the next bit to be transmitted (D0 at this point in the execution) was in the Carry bit. The shifting operation places this bit in the bit 7 position of the accumulator. In addition, the shift places a logic 1 in the bit 6 position from the 80H value that was loaded into the accumulator, thus enabling the *SOD* output line.
- Ⓙ Bit 7 is then output to the *SOD* line and a 1-bit delay is executed. As a result, this second iteration transmits the least significant ASCII character bit (D0).

- Ⓚ The third iteration transmits the D1 ASCII character bit as shown. Likewise, the next six loop iterations transmit the D2, D3, D4, D5, D6, and D7 character bits, in that order. Trace through the algorithm and see if you can verify this transmission.

Notice that with each iteration, the Carry bit is set, shifted into the accumulator, and saved. In the ninth iteration of the loop, the final ASCII character bit is transmitted. At this point, the saved accumulator value will be all 1's. (Why?) So, the last two iterations will transmit two logic 1 STOP bits, right?

Now that you understand the algorithm, the coding is simple. Here's some 8085 code that should do the job:

#### SUBROUTINE SERIAL OUT

OUT	PUSH B	;Save BC on stack
	PUSH D	;Save DE on stack
	PUSH PSW	;Save A and condition flags on stack
	MVI C,0BH	;Initialize C with bit count
	XRA A	;Clear A and Carry
NEXT	MVI A,80H	;Set bit 7 in A
	RAR	;Shift Carry into bit 7 and bit 7 into bit 6
	SIM	;Transmit bit 7 of accumulator
	CALL DELAY	;Delay one bit time
	STC	;Set Carry bit
	MOV A,B	;Load ASCII character in accumulator from B
	RAR	;Shift bit 0 into Carry
	MOV B,A	;Save accumulator in B
	DRC C	;Decrement bit counter (C)
	JNZ NEXT	;Jump to NEXT if all bits not transmitted
	POP PSW	;Restore A and condition flags
	POP D	;Restore DE
	POP B	;Restore BC
	RET	;Return

The above code uses the BC register pair, along with the accumulator and Carry bit to implement the transmit algorithm. The C register is employed as the bit counter. The B register is used to hold the initial ASCII character to be transmitted and then used as a temporary storage register for the accumulator values. The D register is used in the DELAY subroutine to be discussed shortly. You should now have all you need to understand this code.

## SERIAL INPUT

Again, our task will be to develop a general algorithm for serial input. Then we will refine the general algorithm to be applicable to the 8085 MPU, and finally write the 8085 code from the refined algorithm. Here's a general serial input algorithm:

### SUBROUTINE SERIAL IN

```
BEGIN
    Repeat
        Read serial input line
    Until serial input = 0.
    Delay 1/2 bit time.
    Initialize data bit counter to 8.
NEXT   Delay 1 bit time.
        Read serial input line.
        Store bit value.
        Decrement bit counter.
        If bit count not zero then
            Go to NEXT.
        Store the received character in memory.
        Initialize STOP bit counter to 2.
STPBIT Read serial input line.
        If input bit = 0 then
            Call framing error routine.
        Decrement bit counter.
        If bit count not zero then
            Go to STPBIT
END.
```

The first part of the algorithm employs a Repeat/Until loop to read the serial input line until a logic 0 is detected. In other words, the loop is executed until a START bit is detected. Once a START bit is received, a delay of 1/2 bit time is executed so that any subsequent serial read operations will occur in the middle of the bit time period. A counter is then initialized to count the eight character bits (7 ASCII bits + 1 parity bit).

Prior to reading the next bit, a delay of one bit time is executed to assure that the next bit will be read precisely at the middle of its bit time. The bit is read, the counter decremented and tested for zero, and the NEXT loop is repeated until all the character bits have been input. The received character value is then stored in memory for further processing.

The next task is to read the two STOP bits. To do this, the algorithm simply initializes a counter to two and reads the serial input line twice. With each read operation, the input is tested for a logic 0. A logic 0 in a STOP bit position would indicate a **framing** error and the appropriate error routine is called. The subroutine ends when both STOP bits have been successfully read.

Now that you are familiar with the general receive process, let's refine the above algorithm to make it more applicable to microprocessor software in general, and the 8085 in particular. This general algorithm will need to be altered slightly to make is compatible with the 8085 architecture.

#### SUBROUTINE SERIAL IN

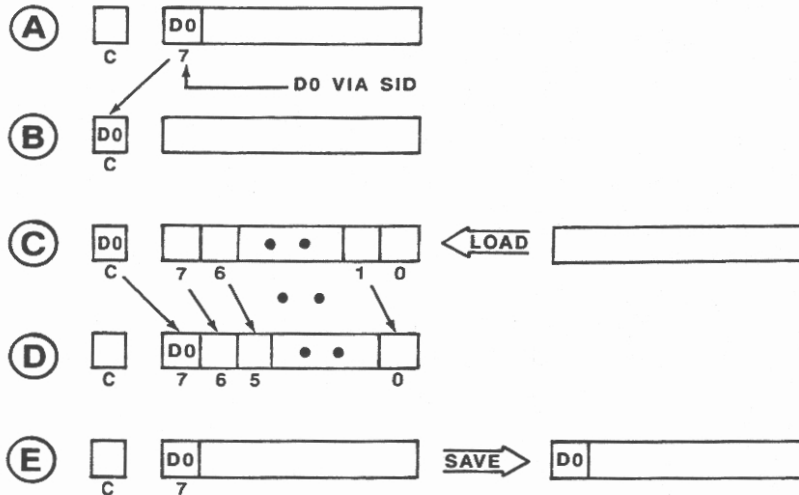
##### BEGIN

```

READ    Read SID line.
        Shift bit 7 position of accumulator left into Carry.
        Jump if Carry not zero to READ.
        Delay 1/2 bit time.
        Initialize a counter register to count eight bits.
NEXT    Delay 1 bit time.
        Read SID line.
        Shift bit 7 of accumulator left into Carry.
        Load previously save bits into accumulator.
        Shift Carry into bit 7 position of accumulator and all other accumu-
        lator bits right one position.
        Save bits received so far in another register.
        Decrement bit counter.
        Jump if not zero to NEXT.
        Store accumulator to memory.
        Initialize STOP bit counter to two.
STPBIT  Read SID line.
        Shift bit 7 of accumulator left into Carry.
        Jump if zero to ERROR.
        Decrement STOP bit counter
        Jump if not zero to STPBIT.
END.
```

Let's now desk check this algorithm to make sure it works. Look at Figure 4-11. We will assume that a START bit has been received and we are about to input the first ASCII character bit (D0). Refer to the keyed references in Figure 4-11 for the following discussion:

NEXT



NEXT

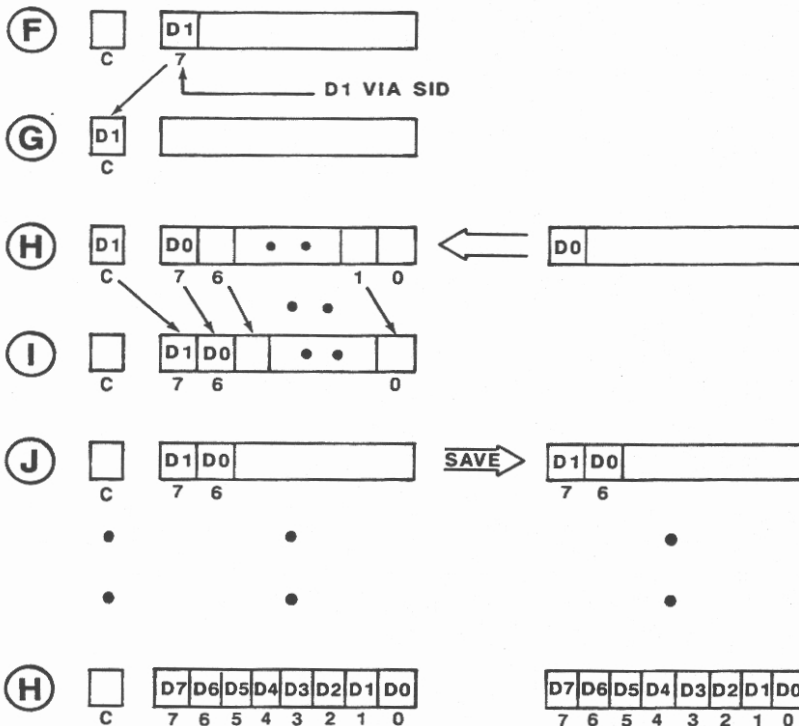


Figure 4-11  
Execution of the SERIAL IN algorithm.

- Ⓐ The *SID* line is read and the received bit is placed in the bit 7 position of the accumulator.
- Ⓑ The received bit is shifted left into the Carry bit.
- Ⓒ Any previously saved bits are loaded into the accumulator. Note that with this first iteration, there have been no ASCII bits previously received.
- Ⓓ The Carry bit is shifted right into the bit 7 position of the accumulator and all other accumulator bits are shifted right one position. This places the received bit (D0) in the bit 7 position of the accumulator.
- Ⓔ The accumulator contents are saved in another CPU register.
- Ⓕ The loop then repeats, and after a one bit delay, the *SID* line is read again. This places the next ASCII bit (D1) in the bit 7 position of the accumulator.
- Ⓖ The received bit (D1) is shifted left into the Carry bit.
- Ⓗ The previously received bits are loaded into the accumulator. Note that in this second loop iteration, the previously received D0 bit is loaded into the bit 7 position of the accumulator.
- Ⓘ The D1 bit just received is shifted right into the accumulator bit 7 position, and the previously received D0 bit is shifted right into the bit 6 position of the accumulator.
- Ⓙ The accumulator contents are saved in another CPU register.
- Ⓚ After six more iterations, the accumulator will contain the received ASCII character with all the received bits in their respective parallel bit positions. In addition, the accumulator is saved in memory for future access to the received character.

The two STOP bits are received and tested but not saved. If a STOP bit is a logic 0, a framing error has occurred and the respective error routine is called. The routine ends when both logic 1 STOP bits have been received.

Now that you know how the process must work within the CPU, we need to implement the above algorithm with 8085 assembler code. Here it is:

#### SUBROUTINE SERIAL IN

IN	PUSH B	;Save BC on stack
	PUSH D	;Save DE on stack
	PUSH PSW	;Save A and condition flags on stack
READ	RIM	;Read <i>SID</i> line
	RAL	;Shift received bit left into Carry
	JC	;Jump if set to READ
	MVI D,TIME/2	;Load D with 1/2 bit TIME value
	CALL HDEL	;Call 1/2 bit delay routine
	MVI C,08H	;Initialize bit counter eight
NEXT	CALL DELAY	;Call full bit delay routine
	RIM	;Read <i>SID</i> line
	RAL	;Shift received bit into Carry
	MOV A,B	;Load previously save bits into A
	RAR	;Shift received bit into bit 7 of A and all other bits right one position
	MOV B,A	;Save received bits in B
	DRC C	;Decrement bit counter
	JNZ NEXT	;Jump not zero to NEXT
	STA 8000H	;Store received character in memory 8000H
	MVI C,02H	;Initialize STOP bit counter
STPBIT	RIM	;Read <i>SID</i> line
	RAL	;Shift received bit left into Carry
	JNC ERROR	;Jump carry zero to ERROR
	DRC C	;Decrement bit counter
	JNZ STPBIT	;Jump not zero to STPBIT
	POP PSW	;Restore A and condition flags
	POP D	;Restore DE
	POP B	;Restore BC
	RET	;Return

Here, the C register has been employed as a bit counter and the B register used as a temporary register to save any previously received bits during the NEXT loop execution. The D register is employed to hold a delay value that is used to create the bit time delay. This timing routine will be discussed shortly. Memory address 8000H will contain the received ASCII character after the routine returns to the calling program. Verify that the code performs the serial receive operation by comparing it to the above algorithm.



## Serial I/O Timing

There are many applications where software time delays must be executed to control I/O operations. You have just observed one such application where time delays are required during the serial input and output routines to establish the serial baud rate. For now, the only way you have to create such delays is to write a subroutine that causes the MPU to loop for a predetermined amount of time. Of course, this wastes MPU time, since it is simply "spinning its wheels" in order to create the time delay. In the next unit, you will learn about programmable timers, whose purpose is to relieve the MPU from such timing tasks and free it up for more important system-related tasks. But, not knowing about these devices at this point requires us to write a software time delay subroutine to generate the bit time delays required in the SERIAL OUT and SERIAL IN subroutines just developed.

Back in the SERIAL OUT subroutine you saw that the bit time period is the amount of time between successive executions of the SIM instructions. On the other hand, the bit time period in the SERIAL IN subroutine is the amount of time between the execution of successive RIM instructions. Given a serial I/O application, both bit times must be the same.

In both the SERIAL OUT and SERIAL IN subroutines, you observed a call to the DELAY subroutine. Here's a simple 8085 looping routine that will create a delay:

### SUBROUTINE DELAY

```
DELAY  MVI D,TIME
HDEL   DCR D
        MOV A,E
        RET
```

Here, the D register is decremented down to zero to create the delay. What controls the amount of time delay? Of course, the number of times the loop is executed! How many times is the loop executed? That depends on the initial value of the D register. If a call is made to DELAY, the D register is initialized with the value TIME. However, if a call is made to HDEL, the D register is decremented from the value TIME/2 which was moved into the D register just prior to the call within the SERIAL IN routine. In other words, a call to DELAY will provide a delay of one bit, while a call to HDEL will generate a 1/2 bit delay.

Now, the question is: What value must TIME be in order to provide a given serial baud rate. To determine a value for TIME, we need to know four things:

- 1) The serial baud rate required by the application.
- 2) The MPU clock frequency.
- 3) The number of MPU states executed between successive SIM and RIM instructions within the SERIAL OUT and SERIAL IN subroutines, respectively.
- 4) The number of MPU states executed within the DELAY subroutine.

The first two items will usually be known. Let's assume that the application requires a 1200 baud rate and that the MPU clock frequency is approximately 1 MHz. A 1200 baud rate means that the total delay required between successive SIM and RIM instructions must be  $1/1200$  bits per second, or  $833 \mu\text{s}$  (microseconds) per bit. Because the clock frequency is 1 MHz, a single MPU state (period) requires  $1/1 \text{ MHz}$ , or  $1 \mu\text{s}$  per state. Now, with a bit time of  $833 \mu\text{s}$ , and an MPU state of  $1 \mu\text{s}$ , there are  $833 \mu\text{s} \times 1 \text{ MPU state per } \mu\text{s}$ , or 833 MPU states required to create a time delay equivalent to one bit time period at 1200 baud.

Our next task is to determine the number of MPU states between successive SIM and RIM instruction executions. From the 8085 data sheet you can determine the following:

SERIAL OUT		MPU States
NEXT	MVI A,80H	7
	RAR	4
	SIM	4
	CALL DELAY	18
	STC	4
	MOV A,B	4
	RAR	4
	MOV B,A	4
	DCR C	4
	JNZ NEXT	10
	Total	<u>63</u>

SERIAL IN		MPU States
NEXT	CALL DELAY	18
	RIM	4
	RAL	4
	MOV A,B	4
	RAR	4
	MOV B,A	4
	DCR C	4
	JNZ NEXT	10
	Total	<u>52</u>

There are 63 MPU states required for the SERIAL OUT routine and 52 states for the SERIAL IN routine. This is a difference of 9 MPU states, representing a time period of  $9 \times 1 \mu\text{s}$ , or  $9 \mu\text{s}$ . This slight difference should not affect the timing for this application. If it did, we could insert a couple of dummy instructions, like ORI 0H, in the SERIAL IN routine so that each routine had the same number of MPU states. So let's assume that each routine requires 63 MPU states to execute.

Now for the delay routine analysis. Here it is:

	MPU States	Number of Times Executed
DELAY MVI D,TIME	7	Once
HDEL DCR D	4	TIME times
JNZ HDEL	10	TIME times
RET	10	Once

Here you see that two of the instructions are only executed once (MVI D and RET). These two instructions require 17 MPU states. If you add this value to the number of MPU states required by the SERIAL OUT routine you get  $63 + 17$ , or 80 MPU states. This leaves  $833 - 80$ , or 753 MPU states that must be executed by the loop in the DELAY routine. Since each loop within the DELAY routine uses  $4 + 10$ , or 14 states, the loop must be executed  $753/14$ , or 54 times to create the 753 required MPU states. Thus the value of TIME must be 54 decimal, which is equivalent to 36 hex (H). This value must be inserted in the DELAY machine code for TIME. In addition, the value  $36\text{H}/2$ , or 1CH, must be inserted in the SERIAL IN machine code for the value TIME/2.

## Hardware Conversion

The advantage of software conversion is its hardware simplicity. With the 8085, no additional hardware components are required to provide serial I/O directly from the MPU. However, the MPU must execute the required software routines. As the system gets more complex, such tasks tie-up the MPU when its time could be better spent performing more critical system chores. In addition, you found that there is a speed limitation to the serial I/O process due to the MPU execution speed. For these reasons, many microcomputer systems employ hardware parallel/serial conversion. You discovered with software conversion that the process is simply a register shifting operation. The same is true of hardware conversion. The hardware devices which provide the conversion are simply controlled, or programmable, shift registers.

There are three generic names for hardware devices which provide parallel/serial conversions. They are the **Universal Asynchronous Receiver-Transmitter (UART)**, **Universal Synchronous Receiver-Transmitter (USRT)**, and **Universal Synchronous/Asynchronous Receiver-Transmitter (USART)**. As you can tell from their names, the UART is used with asynchronous serial data, the USRT is used with synchronous serial data, while the USART can be used with either asynchronous or synchronous serial data. The UART is generally used in microcomputers for low-to-medium speed data transmission. The USRT is used in high speed data transmission applications.

The internal structure of these devices can be functionally divided into three major sections: a **transmit section**, a **receive section**, and a **status/control section**. These three functional sections are shown in Figure 4-12.

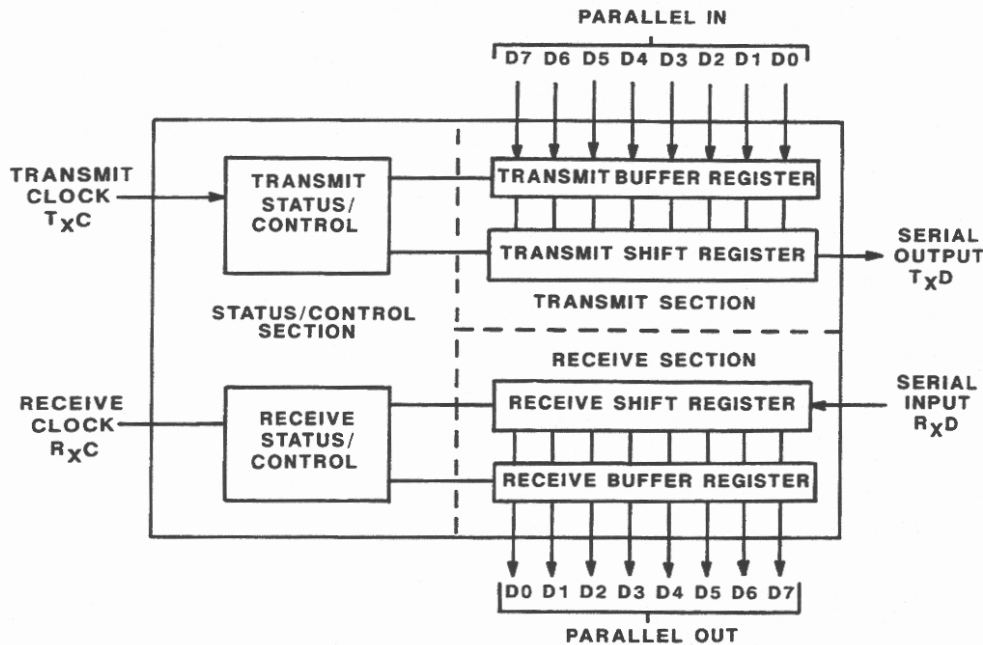


Figure 4-12

The functional structure of a typical UART.

Parallel data to be converted to serial is written by the MPU to the **transmit buffer register**. The parallel data is then transferred to the **transmit shift register** and shifted out the serial output line at a rate determined by the transmit clock and control section. The **START**, **STOP**, and parity bits are added automatically by the device for asynchronous transmission.

Serial data to be converted to parallel will enter the serial input line into the **receive shift register**. The serial data is shifted into the receive shift register at a rate determined by the receive clock and control section. The START and STOP bits are automatically removed from the asynchronous character and the received character value is transferred to the parallel output **receive buffer register** where it can then be read by the MPU.

The status/control section not only controls the transmission/reception rate, but has many additional status and control functions. It is used to generate interrupts, check parity, framing, etc. Each of these status/control functions will be discussed in the next section.

General purpose UARTs, USRTs, and USARTs were some of the first LSI devices to be standardized. However, since the dawning of the microprocessor, most manufacturers have developed devices that are directly compatible with their microprocessor families. In the Motorola 6800 family, the 6850 Asynchronous Communications Interface Adapter, or ACIA, is the device which is used to provide asynchronous communication. In the Intel 8085/8088/8086 family, the 8256 Multifunction Universal Asynchronous Receiver-Transmitter, or MUART, is used to provide asynchronous communication.

You have already been acquainted with the parallel I/O and interrupt features of the 8256 MUART. It is now time to learn about the asynchronous serial I/O function of the MUART. This is the topic of the next section.

## Self-Test Review

14. List the two serial I/O lines available on the 8085 along with their corresponding 8085 instructions.
15. Serial data is transmitted and received via bit \_\_\_\_\_ of the \_\_\_\_\_ within the 8085 MPU.
16. What major software or hardware operation is involved when making parallel/serial conversions?
17. What bit within the 8085 accumulator must be set in order to enable the serial output data line?
18. Why must a 1/2 bit time delay be created after receiving a START bit in the SERIAL IN routine?
19. Which status flag within the MPU is used as a temporary serial bit storage location for both the SERIAL OUT and SERIAL IN routines?
20. A \_\_\_\_\_ error is when the proper number of bits are not present in the asynchronous transmission.
21. Why must an ORing operation be performed to test for zero when decrementing a register pair to create a time delay?
22. What is the difference between a call to DELAY versus HDEL within the SERIAL IN subroutine?
23. What four things must be known in order to calculate an initialization value for a bit time delay routine?
24. What must be the value of TIME in the DELAY subroutine in order to create the required time delay for a 4800 baud rate using an MPU clock frequency of 1 MHz?
25. Based on your calculation for 4800 baud in Question 24, is there a problem if the baud rate is increased to 9600?
26. What are the advantages and disadvantages of software conversion?
27. Three standard LSI devices which provide parallel/serial conversions are the \_\_\_\_\_, \_\_\_\_\_, and \_\_\_\_\_.

28. The internal structure of the above three devices can be divided into what three functional sections?
29. A programmable parallel I/O device that also has an internal UART and is compatible with the Intel 8085/8088/8086 family of MPUs is the \_\_\_\_\_.

## Answers

14. The 8085 *SOD* line transmits serial data using the *SIM* instruction. The 8085 *SID* line receives serial data using the *RIM* instruction.
15. bit 7, accumulator
16. Shifting.
17. Bit 6.
18. A 1/2 bit time delay is created after receiving a *START* bit so that the subsequent bits are received in the middle of the bit period. This reduces the possibility for error.
19. The Carry bit.
20. framing
21. An ORing operation must be performed to test for zero when decrementing a register pair because the 8085 *DCX* instruction does not affect the Zero flag in the MPU.
22. A call to *DELAY* creates a full bit delay, while a call to *HDEL* creates a 1/2 bit time delay.
23. The serial baud rate, the MPU clock frequency, the number of MPU states between successive *SIM* and *RIM* instructions, and the number of MPU states within the delay subroutine.
24. 09H.
25. Yes, the value of *TIME* is too small to create an accurate time delay. The only way to compensate is to increase the MPU clock frequency or use hardware conversion.
26. The advantage of software conversion is hardware simplicity. The disadvantages are limited transmission speed and tying-up the MPU.
27. *UART*, *USRT*, and *USART*.
28. The internal structure of the above devices can be divided into a transmit, receive, and status/control section.
29. 8256 Multifunction Universal Asynchronous Receiver-Transmitter, or *MUART*.



## SERIAL I/O USING THE 8256 MUART

Another feature of the 8256 MUART is to provide asynchronous serial I/O for the 8085/8088/8086 family of microprocessors. When performing parallel/serial conversions, a device like the MUART frees the MPU from the software required to make the conversion and allows it to handle more important tasks. When transmitting asynchronous data, the MUART will automatically add the required START, STOP, and parity bits. When receiving asynchronous data, the MUART will automatically strip off the START and STOP bits. In addition, it will check for proper parity and framing of the asynchronous input character.

Several data formats and parity options are software selectable by programming the MUARTs command registers. Also, several baud rates are software selectable. Once the MUART is initialized, you simply write to a port to transmit serial data and read from the same port to receive serial data. Of course, this assumes that the MUART is connected to the MPU for direct I/O.

In this section, you will be introduced to the serial I/O feature of the MUART. You will learn how to initialize it for various asynchronous applications as well as to communicate with it in order to perform the serial I/O experiment that follows.

### Functional Layout and Registers

The I/O lines and registers associated with the serial I/O section of the MUART are shown in Figure 4-13. Parallel data from the MPU is written to the **Transmit Buffer Register** of the MUART. From there, the MUART transfers the data to the **Transmit Shift Register** to be shifted out on the *TxD* transmit data line. In order for the shifting operation to take place, the MUART Clear to Send ( $\overline{CTS}$ ) line must be low. This line enables the transmit shift register.

Serial data from a peripheral device is input via the *RxD* receive data line and shifted into the **Receive Shift Register**. When all the bits have been received, the MUART transfers the received character to the **Receive Buffer Register** where it can be read by the MPU. Reading the Receive Buffer Register gates its contents onto the MPU data bus.

The *TxC* transmit clock and *RxC* receive clock lines control the rate at which the shift registers are clocked, thereby controlling the serial baud rate. External clocking signals can be applied to these lines to control the baud rate, or the shifting operation can be clocked from the internal MUART clock frequency which is connected to the MPU clock line.

The MUART *INT* and  $\overline{INTA}$  lines can be employed for interrupt control of the serial transmit and/or receive operations. This will be discussed shortly.

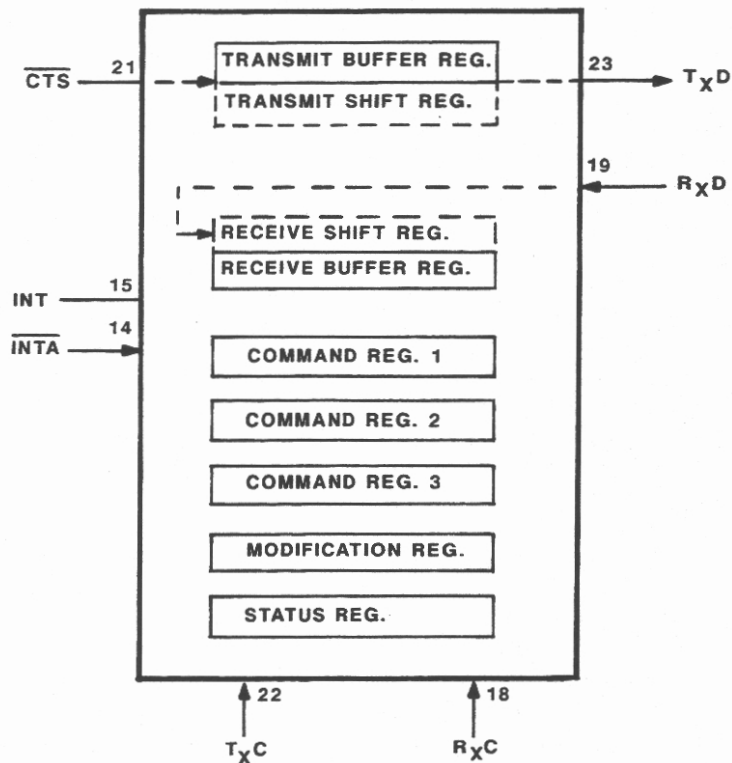


Figure 4-13

The serial I/O section of the MUART.

## Registers and Addressing

As stated earlier, Figure 4-13 shows those registers within the MUART that are associated with serial I/O. All these registers are user-accessible, except for the Transmit Shift Register and the Receive Shift Register. The purpose of these two is to simply shift the serial word in or out, once it has been applied to the MUART. Let's discuss the remaining registers in detail, since they are all user-accessible and must be employed when performing serial I/O with the MUART.

## TRANSMIT BUFFER REGISTER

This register is a *write-only* register assigned to port address X8. From Figure 4-14, you see that a write (OUT) operation to this port address loads the Transmit Buffer Register with the current parallel data on the data bus. From there, it is transferred to the Transmit Shift Register and shifted out as serial data on the  $TxD$  line.

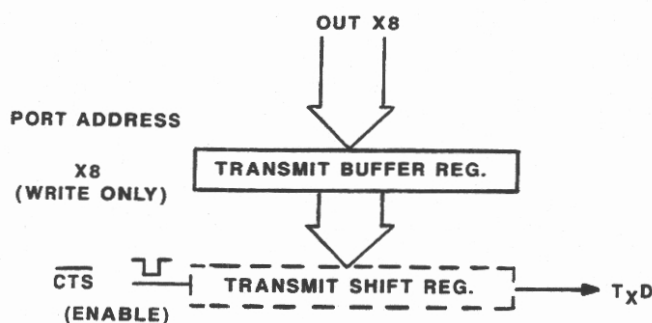


Figure 4-14

Parallel data written to the Transmit Buffer Register is passed to the Transmit Shift Register for serial transmission when  $\overline{CTS}$  is enabled low.

For instance, the instruction OUT X8 will transfer the contents of the MPU accumulator to the Transmit Buffer Register. Here, X is the decoded port address for the MUART, while 8 is the address of the Transmit Buffer Register within the MUART.

Once a parallel value is written to the Transmit Buffer Register, it is transferred to the Transmit Shift Register only if the Clear To Send ( $\overline{CTS}$ ) line on the MUART is low. In other words, the  $\overline{CTS}$  line must be low in order for the character in the Transmit Buffer Register to be transmitted from the MUART. The  $\overline{CTS}$  line can be tied low, or the MPU can pulse the line each time a character is written to the Transmit Buffer Register.

## RECEIVE BUFFER REGISTER

The Receive Buffer Register is a *read-only* register at port address X8. When a serial character is received on the *RxD* line, it is automatically shifted into the Receive Shift Register and then transferred to the Receive Buffer Register. Once in the Receive Buffer Register, the character can be read by the MPU at port address X8. Thus, the instruction *IN X8* will load the accumulator with the parallel equivalent of a serial input character that has been received by the MUART.

As shown in Figure 4-15, the Receive Buffer Register receives the parallel equivalent of a serial input from the Receive Shift Register. This value is gated onto the MPU data bus when the Receive Buffer Register is read by the MPU. Note that there is *no* special enabling line on the MUART required to transfer data from the Receive Shift Register to the Receive Buffer Register as there is for the transmit operation.

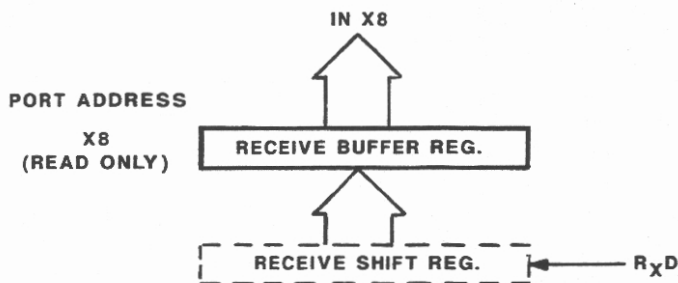


Figure 4-15

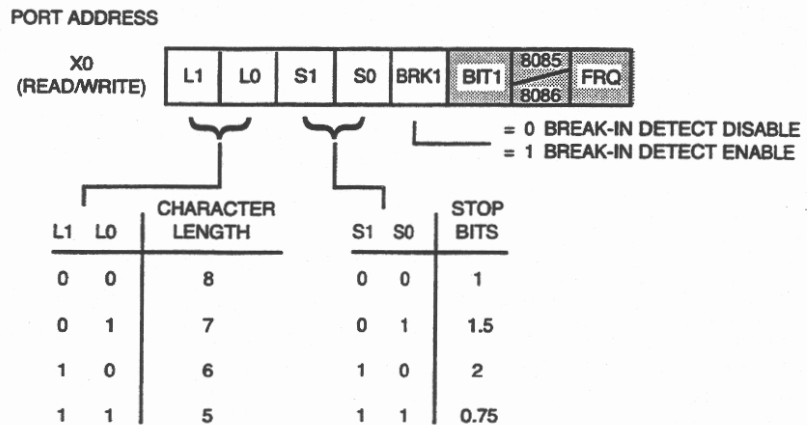
Serial input data is shifted into the Receive Shift Register and transferred to the Receive Buffer Register where it can be read by the MPU.

## COMMAND REGISTER 1

Command Register 1 is a read/write register located at port address X0. The upper five bits of this register are used to control three serial I/O characteristics within the MUART as follows:

- Bits L1 and L0 control the serial character length.
- Bit S1 and S0 control the number of stop bits.
- Bit BRKI enables the break-in detect feature.

The various bit combinations and corresponding control options for these bits are summarized in Figure 4-16. We will begin our discussion with the L1, L0 bits.



**Figure 4-16**

Command Register 1 is used to set the character length, stop bits, and break-in detect feature.

### Character Length (Bits L1, L0)

Notice that the logic present in the L1, L0 bits determines the serial character length. You have the option of a 5-, 6-, 7-, or 8-bit character length. You will typically configure the MUART for a 7- or 8-bit character length.

### Stop Bit Length (Bits S1, S0)

The logic present in the S1, S0 bits determines the number of stop bits that are automatically added to a serial transmission, and detected during serial reception. From Figure 4-16 you see that you have the option of 1, 1.5, 2, or .75 stop bits. You will typically configure the MUART for 1 or 2 stop bits, depending on the peripheral requirements.

## Break-In Detect Enable (BRKI bit)

The BRKI bit is called the Break-In Detect Enable bit. The break-in feature is used with half-duplex communication. Recall that half-duplex is two-way communication on one line. Thus, the TxD and RxD lines of the MUART must be tied together for half-duplex operation. Now, suppose that the MUART is transmitting a character to some serial peripheral device and the peripheral device needs to interrupt, or "break", the character transmission stream. Maybe, the peripheral needs to send a character back to the MUART, or simply break the transmission for a period of time.

The later case often occurs, for example, with a serial printer. The MPU transmits a series of characters to the printer. However, the MPU can transmit characters much faster than the printer can print them. To compensate for the speed difference, the printer stores a fixed number of the received characters in a print buffer while they are being printed. When the MPU fills up the print buffer, the printer must tell the MPU to break the transmission until the print buffer is empty. Thus, the printer must generate a break to the MPU to tell it to stop sending characters.

Here's how the break-in feature works within the MUART. When the peripheral receives a stop bit from the MUART, indicating the end of the current character, it can generate a space on the half-duplex line to signal the break condition. If the break-in feature is enabled via the BRKI bit in Command Register 1, the space received from the peripheral is detected and the MUART "breaks" its transmission.

To use the break-in feature, you must connect the *TxD* pin to the *P16* pin on the MUART. To detect a break-in, the MUART monitors the logic on the *P16* line during the transmission of the last stop bit. If the *P16* line goes to a logic 0 state any time during the last stop bit, a break from the peripheral has been detected and the MUART ceases its transmission. Of course, this assumes that the break-in feature is enabled by setting the BRKI bit in Command Register 1. In addition, the MUART does not automatically cease transmission upon detecting a break. The transmission software must detect the break via the MUART Status Register. Once a break is detected, the software ceases the transmission by not writing additional characters to the Transmit Buffer Register or by toggling the Clear To Send (*CTS*) line on the MUART high to disable the transmitter section of the MUART. This will be covered in more detail when we discuss the Status Register shortly.

## COMMAND REGISTER 2

Command Register 2 is a *read/write* register located at port address X1. All the bits within Command Register 2 are associated with the serial I/O feature of the MUART in one way or another. Let's begin our discussion of this register with the lower four bits, labeled B3 - B0, in Figure 4-17.

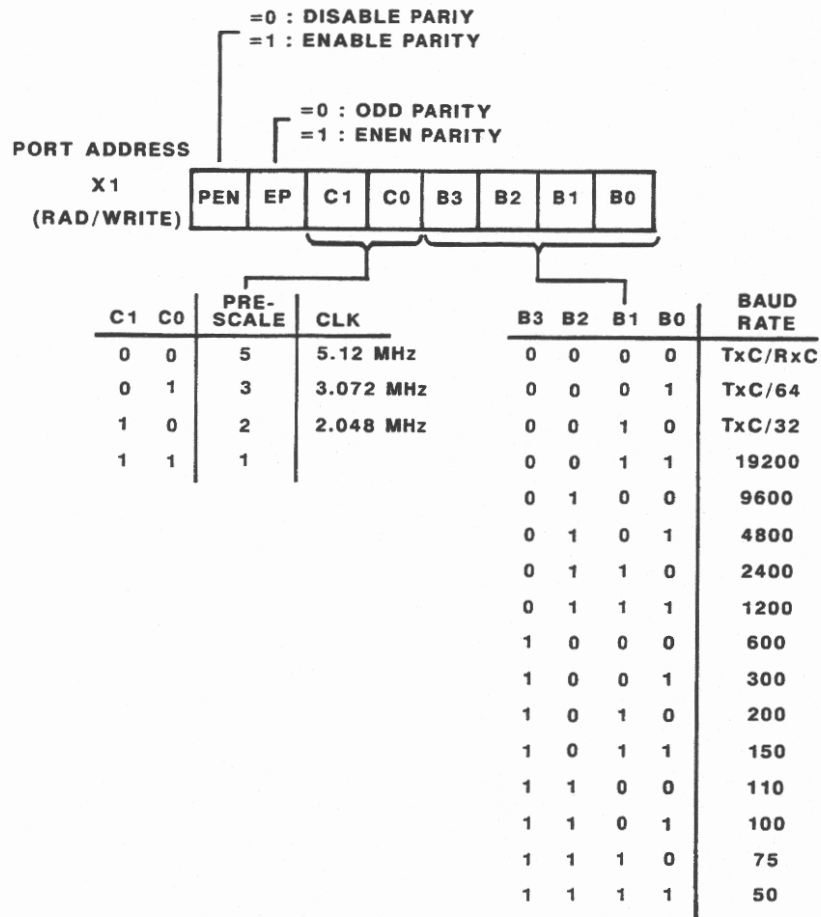


Figure 4-17

Command Register 2 is used to set parity, the internal MUART clock frequency, and the serial baud rate.

## Baud Rate Select (Bits B3 - B0)

Bits B3 - B0 are called the baud rate select bits because they are used to select the serial I/O baud rate. The baud rate can be derived from the *TxC* or *RxC* clock input lines on the MUART, or the internal MUART clock frequency which is derived from the *CLK* input to the MUART. The table in Figure 4-17 shows the baud rate options available from the logic present in these four bits. There are basically three baud rate modes:

- 1) When B3 - B0 are all reset to a logic 0 state (0H), *separate* clock frequencies can be input on the *TxC* and *RxC* serial clock input lines of the MUART. Thus, two different baud rates can be used for transmission and reception, respectively. The respective frequencies input on the *TxC* and *RxC* input lines must equal the desired baud rate. For example, to transmit at 9600 baud and receive at 4800 baud, you must apply a 9600 Hz signal to the *TxC* pin and a 4800 Hz signal to the *RxC* pin.
- 2) Programming B3 - B0 with the hex value 1H (0001) or 2H (0010), defines the *TxC* clock input line as the common clocking source for both the transmitter and receiver. In this mode, the *TxC* signal must be 64 or 32 times the desired baud rate. The signal must be 64 times the desired baud rate when B3 - B0 are programmed with a 1H, and 32 times the desired baud rate when programmed with a 2H. For instance, to establish a 9600 baud rate, bits B3 - B0 must be programmed with a 1H and the *TxC* input signal must be  $64 \times 9600$ , or 614.4 KHz. Of course, you could also program bits B3 - B0 with a 2H and apply a  $32 \times 9600$ , or 307.2 KHz signal to the *TxC* line to achieve a 9600 baud rate. You should be aware, however, that the *TxC* input signal cannot exceed 1.024 MHz.
- 3) Programming the B3 - B0 with values from 3H (0011) through FH (1111) enables the internal MUART baud rate generator. In these modes, the serial baud rate is derived internally from the internal clock frequency of 1.024 MHz, rather than the *TxC* or *RxC* input lines. You can see from the table in Figure 4-17 that baud rates of 19200 down to 50 are available by programming B3 - B0 with values from 3H through FH, respectively.



### Internal Clock Prescaler (Bits C1, C0)

The internal MUART clock frequency must be 1.024 MHz. However, you can apply a x1, x2, x3, or x5 signal at the *CLK* pin of the MUART to obtain this internal operating frequency. To obtain 1.024 MHz from a x5, or 5.12 MHz CLK, frequency you must program the C1, C0 bits with logic 0s. On the other hand, if a 1.024 MHz signal is applied to the *CLK* input line, the C1, C0 bits must be set to logic 1s to obtain a x1, or 1.024 MHz, internal clock frequency. The C1, C0 table in Figure 4-17 summarizes these CLK options.

### Even Parity (EP bit)

The Even Parity bit (EP) is used to select the type of parity, even or odd. When EP is set to a logic 1 state, even parity is selected, while odd parity is selected when the EP bit is reset to a logic 0 state.

### Parity Enable (PEN bit)

The Parity Enable bit (PEN) must be set in order to provide for parity with serial I/O. When the PEN bit is set, the MUART will insert a parity bit between the last character bit and the first stop during transmission. In addition, the parity is checked during reception. If improper parity is detected during reception, a bit is set within the MUART Status Register. This will be discussed shortly. Of course, if PEN is reset to a logic 0 state, no parity generation or checking is provided.

**Example 4-1:** Write the 8085 code required to configure the MUART for serial I/O as follows:

- Seven character bits.
- Odd parity.
- One stop bit.
- 1200 baud for both transmit and receive, derived from the internal clock frequency of 1.024 MHz.
- Break-in feature disabled.

From Figures 4-16 and 4-17, you determine the following:

Command Register 1 (Figure 4-16)

- Bits L1, L0 must be 01 to select seven character bits.
- Bits S0, S1 must be 00 to select one stop bit.
- The BRKI bit must be cleared to disable the break-in feature.
- The 8085/8086 bit must be cleared to put the MUART in the 8085 operating mode (Ref. Unit 1).
- The BITI bit can be set or reset depending on the option desired (Ref. Unit 1). We will assume it is to be reset (logic 0).
- The FRQ bit can be set or cleared, since it has no effect on the serial I/O operation of the MUART. This bit is used for the timer section of the MUART and will be discussed in Unit 5. For now, we will assume that it is to be a logic 0.

Consequently, Command Register 1 must be programmed with the binary value 0100 0000, or 40H.

Command Register 2 (Figure 4-17)

- The PEN bit must be set to enable the parity feature.
- The EP bit must be cleared to select odd parity.
- The C1, C0 bits will be programmed with 11, assuming the *CLK* line input frequency is 1.024 MHz.
- The B3 – B0 bits will be programmed with 0111 to select 1200 baud from the internal baud rate generator.

Thus, Command Register 2 must be programmed with the binary value 1011 0111, or B7H.

Here is the required 8085 code:

```
MVI A,40H
OUT 50H
MVI A,B7H
OUT 51H
```

Notice that the Command Register 1 and 2 values are moved into the accumulator then output to the respective register addresses. We have assumed that the MUART is decoded for port addresses 50H through 5FH. Thus, Command Register 1 is located at port address 50H and Command Register 2 is located at port address 51H.

## COMMAND REGISTER 3

Command Register 3 is a *read/write* register located at address X2H. The bits in Command Register 3 that are associated with serial I/O are shown in Figure 4-18. Let's begin our discussion with the RST bit, since it also has control over other MUART operations.

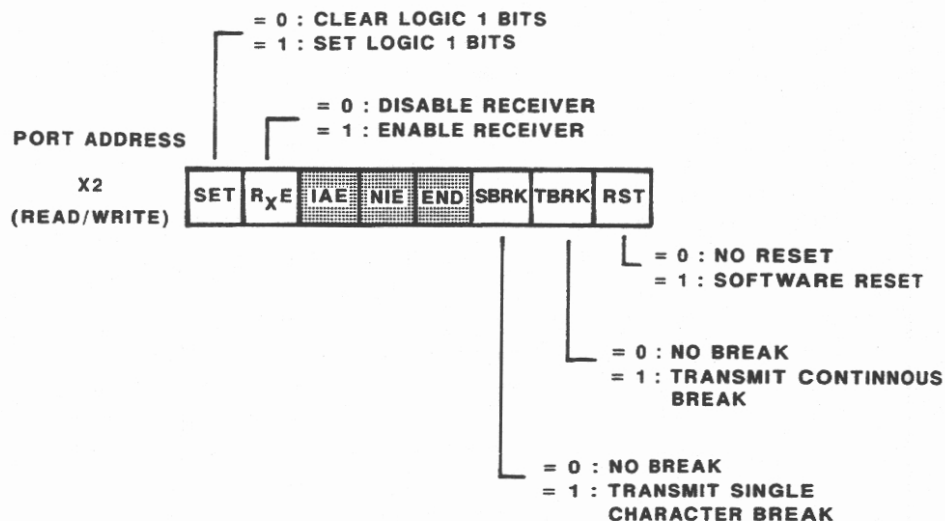


Figure 4-18

Command Register 3 is used for a variety of serial I/O control functions, including a software reset.

### Reset (RST bit)

The RST bit in Command Register 3 provides a way for you to software reset the MUART. When you set the RST bit, the MUART is automatically configured as follows:

- The receiver and transmitter are reset. This means that the transmitter enters an idle mode and the receiver enters a start bit search mode. An idle mode for the transmitter is defined as a mark condition. Thus, the *TxD* line goes to a logic 1 state when RST is set.

- All status bits within the Status Register are cleared, except bit 4 (TRE) and bit 5 (TBE) which are set. The TRE bit and TBE bit within the status register indicate that the two transmit registers are empty. These two bits will be discussed shortly.
- If Port 2 is programmed for the handshake mode, the  $\overline{IBF}$  and  $\overline{OBF}$  control lines of Port 1 are set to a logic 1 state.
- The Interrupt Enable, Set Interrupts, Reset Interrupts, and Interrupt Address registers are all cleared. Any pending interrupts will be canceled and the *INT* interrupt output line of the MUART will go inactive (low).

You should be aware that setting the RST bit does not alter any information in the parallel port registers. In addition, none of the information in any of the three Command Registers is altered. However, any MUART operation in progress is canceled.

There is also a hardware reset available via the *RESET* pin on the MUART. A high level signal on the *RESET* pin activates the hardware reset. The above software reset operations are a subset of the hardware reset, since the hardware reset accomplishes all the above but, in addition, clears Command Registers 1, 2, and 3, the Mode Register, the Port 1 Control Register, and the Modification Register. Consequently, a hardware reset provides a complete re-initialization of the MUART to a well-defined state. You might note that by clearing the Mode Register and Port 1 Control Register both Port 1 and Port 2 are configured as parallel input ports.

## Transmit Break (TBRK bit)

The TBRK bit allows you to transmit a break condition via the *TxD* line under software control. Recall that a break condition is defined as a space, or low, state on the serial line. To transmit a break, you must set the TBRK bit. When the TBRK bit is set, the *TxD* line will go low as soon as any current character transmission is finished. The *TxD* line will remain low until the TBRK bit is reset back to a logic 0. Furthermore, no transfer of data will take place between the Transmit Buffer Register and the Transmit Shift Register until the TBRK bit is reset. In other words, the transmitter section of the MUART is disabled while TBRK is set.

### Single Character Break (SBRK)

Setting the SBRK bits causes the transmitter to generate a single break character. A break character occurs when all bits, including the START bit, data bits, parity bit and STOP bits are spaces. The break character will begin as soon as any existing character transmission is completed and stop after the required number of bit times have passed. Once the break character has been transmitted, the SBRK bit is automatically reset.

### Receive Enable (RxE bit)

This bit is used to enable/disable the receiver section of the MUART. The RxE bit must be set for the MUART receiver to operate. When the RxE bit is cleared, the receiver is disabled and the associated receive bits in the Status Register are not affected by any information on the *RxD* input line.

### Bit Set/Reset (SET)

This bit provides a convenient means to set or reset the rest of the Command Register 3 bits. If SET is high during a write operation to Command Register 3, then any bit that is set will remain set. On the other hand, if SET is low during a write operation to Command Register 3 then any bit that is set will toggle to a low state.

## MODIFICATION REGISTER

The Modification Register is a *write-only* register located at port address XFh. All the bits in this register are associated with serial I/O as shown in Figure 4-19. Let's begin with the least significant bit, DSC.

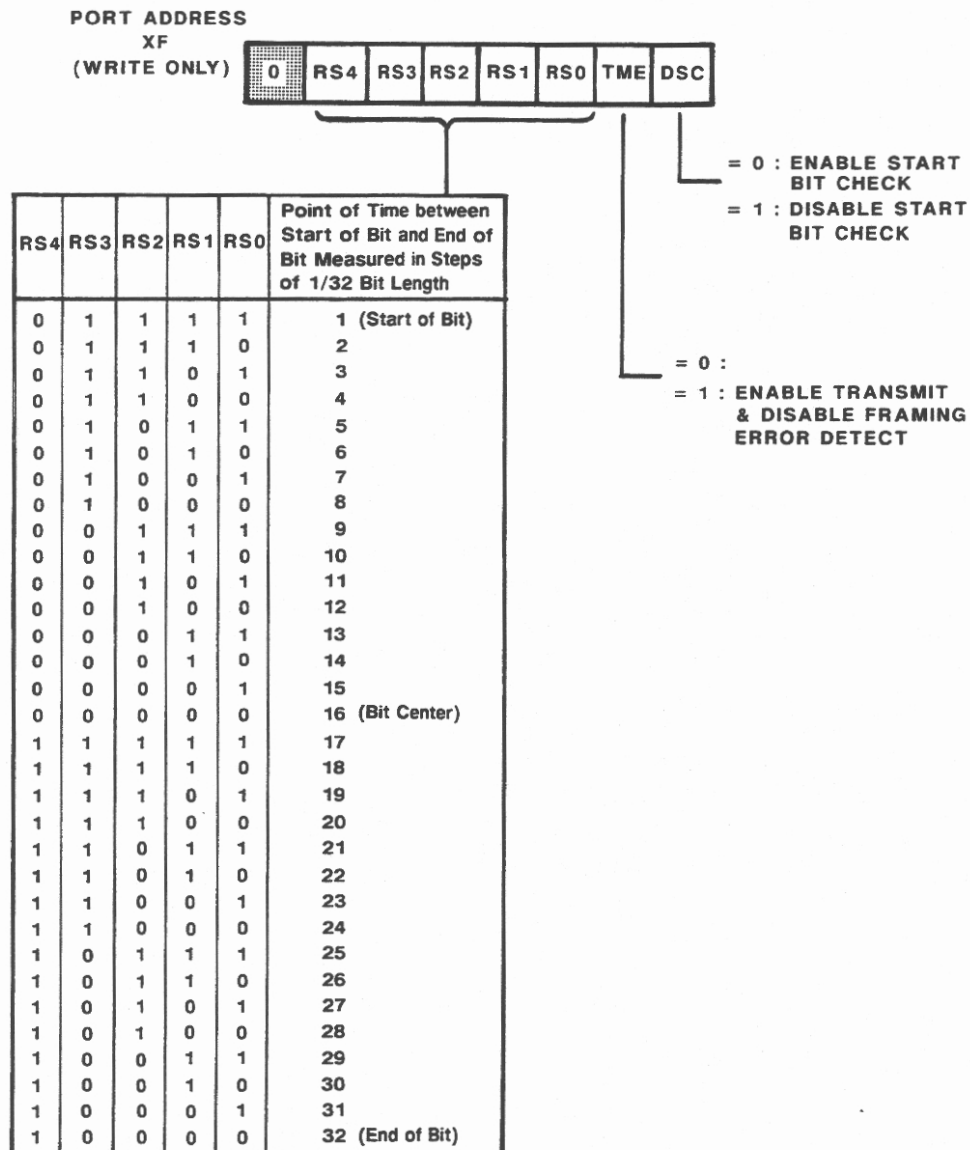


Figure 4-19

The Modification Register is used to set the sampling point of the received bits and to enable/disable start bit and framing features.

### **Disable Start Bit Check (DSC bit)**

When this bit is set, the start bit check is disabled. In other words, no check is made for the start bit during a receive operation. Consequently, the receiver will not be reset to begin receiving a new character if the first bit received is not low. For this reason, you will always want the DSC bit cleared. Recall that a hardware reset clears the Modification Register and, therefore, the DSC bit. However, a software reset does not clear this bit.

### **Transmission Mode Enable (TME bit)**

This bit enables a special transmission mode for the MUART. When TME is set, the MUART can determine if it is receiving its own transmitted character via the FE bit in the Status Register. This feature will be discussed further as you learn about the Status Register.

### **Receiver Sample Time (Bits RS4 - RS0)**

You can modify the sampling point of the receiver by programming the RS4 - RS0 bits in the Modification Register per the table in Figure 4-19. Notice that the sampling points are in increments of 1/32 of the bit length. Thus, a sampling point of 1 will sample the received bits at a point 1/32 into their bit time. A sample point of 2 will sample the bits at a point 2/32 into their bit time, and so on.

For instance, suppose the MUART is configured to receive at 9600 baud. With 9600 baud, the bit time is  $1/9600$ , or 104 microseconds. If RS4 - RS0 were programmed with a binary 00110, the sampling point would be  $10/32 \times 104$  microseconds, or 32.5 microseconds from the start of the bit time period.

You might have noticed from the table in Figure 4-19 that the bits are sampled in the middle of their bit time period when RS4 - RS0 are all cleared. Thus, a hardware reset on the MUART automatically configures the receiver to sample in the middle of the bit time period. This is the optimum sample point.

Finally, you should be aware that the sample point can only be modified using RS4 - RS0 when the internal baud rate generator is being used to establish the baud rate. In other words, the sampling point cannot be altered when the receiver is being clocked from the MUART *RxC* line.

## STATUS REGISTER

As shown in Figure 4-20, the MUART contains a single *read-only* Status Register located at address XF. The most significant Status Register bit is the INT bit which was discussed in Unit 3. The lower six bits of the Status Register are all associated with the serial section of the MUART as follows:

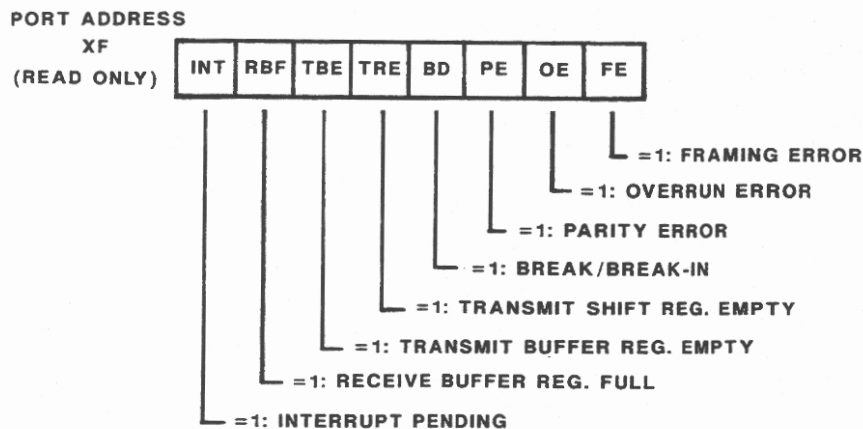


Figure 4-20

The Status Register reflects the status of the serial I/O operations within the MUART.



### **Framing Error (FE bit)**

The FE bit has two purposes: 1) to indicate a framing error, or 2) to indicate that the MUART has received the same character that it transmitted. Normally, the FE bit indicates that a framing error has been detected. A framing error occurs when the receiver does not detect the first stop bit. In other words, a framing error occurs when the stop bit position is received as a space. If this happens, the FE bit will set and the received character will be transferred to the Receive Buffer Register as normal. In addition, the logic 0 stop bit is treated like a start bit and the receiver will assemble the subsequent bits as if they are another character. So, the idea is to check the FE bit in the Status Register before reading a character from the Receive Buffer Register. If set, the FE bit is cleared by reading the Status Register or by a software or hardware reset of the MUART.

The FE bit can also be used to tell when the MUART has received the same character that it transmitted. To do this, the TME bit in the Modification Register must be set. When TME is set, the FE bit in the Status Register will indicate that the character received was transmitted by the MUART. Once set in this mode, only a hardware reset will clear the FE bit. Simply reading the Status Register has no affect on the FE bit.

### **Overflow Error (OE bit)**

An overflow error occurs when a character in the Receive Buffer Register has not been read before the next character is received. If this happens, the first character in the Receive Buffer Register is lost. The overflow condition is indicated when the OE bit is set. If set, the OE bit is cleared by reading the Status Register or by a software or hardware reset of the MUART.

### **Parity Error (PE)**

A parity error occurs and the PE bit sets when improper parity is received. Recall that the correct parity (even or odd) is determined by the EP (Even Parity) bit in Command Register 2. In addition, the parity check feature must be enabled by the PEN (Parity Enable) bit in Command Register 2.

Once set, the PE bit is cleared by reading the Status Register or by a software or hardware reset of the MUART.

## Break/Break-In (BD)

The BD bit is set when either a break character (all spaces) has been received on the *RxD* line or when a break-in condition is detected on the *TxD* line. A break character is when all bits in the asynchronous word, including the first STOP bit, are spaces. When a break character is received on the *RxD* line, the receiver is idled and the received break character is not transferred to the Receive Buffer Register. The receiver is started again with the next high-to-low transition on the *RxD* line, indicating the START bit of a new character. The break condition is detected even when the receiver is not enabled. Thus, a break character can be detected at any time.

If the break-in detect feature is enabled via the BRKI bit in Command Register 1, the BD bit in the Status Register will also set if a break-in is detected on the *TxD* line. Recall that a break-in on the *TxD* line is when a peripheral generates a START bit during half-duplex operation right after the transmitter has transmitted a STOP bit.

Suppose the BD bit is set and the transmit break-in feature is enabled via the BRKI bit in Command Register 1. How do you determine if the BD bit reflects a received break character on the *RxD* line or a break-in condition on the *TxD* line? Simple, the MUART generates a Level 4 receive interrupt when a break character is received on the *RxD* line, while a Level 5 transmit interrupt is generated when a break-in occurs on the *TxD* line. Of course, these interrupts must be enabled via the Interrupt Enable register, otherwise they must be polled as discussed in Unit 3.

## Transmit Shift Register Empty (TRE)

The TRE bit sets when the Transmit Shift Register is empty. In addition, a Level 5 transmit interrupt is generated if enabled. Conversely, the TRE bit is a logic 0 if the transmitter is in the process of transmitting a character. The TRE bit is set by the transmission of the last STOP bit in a character and also by a software or hardware reset of the MUART.

### **Transmit Buffer Register Empty (TBE)**

The TBE bit sets when a character has been transferred from the Transmit Buffer Register to the Transmit Shift Register. Thus, the TBE bit indicates that the Transmit Buffer Register is empty and the next character can be written to the MUART for serial output. You will use this bit to indicate when a character can be written to the Transmit Buffer Register.

A Level 5 transmit interrupt is also generated, if enabled, when the TBE bit sets. So how do you tell whether a Level 5 interrupt is due to the Transmit Buffer Register being empty or the Transmit Receive Register empty? Easy, read the Status Register to see which bit is set (TRE or TBE). One or the other, not both, must cause the interrupt to be generated, right? Think about it!

Finally, the TBE bit is automatically set with a software or hardware reset.

### **Receive Buffer Register Full (RBF)**

The RBF bit sets when a character is transferred from the Receive Shift Register to the Receive Buffer Register, indicating that the Receive Buffer Register is full and ready to be read by the MPU. Thus, you will use this bit to indicate when a character can be read from the Receive Buffer Register. When RBF sets, a Level 4 receive interrupt is generated by the MUART, if enabled. The RBF bit is cleared when the Receive Buffer Register is read and also by a software or hardware reset of the MUART.

### **Interrupt Pending (INT bit)**

This bit was discussed in Unit 3. Recall that the INT bit sets whenever any interrupting source is active. It is usually used in connection with polled control of I/O. You can control the serial I/O process without using interrupts by polling the INT bit in the Status Register. If the INT bit is set, the other status bits in the Status Register can be checked to determine the service needed. However, interrupts are much more efficient and are used almost exclusively to control serial I/O. This is especially true when you have the interrupt handling capabilities of device like the MUART. So, let's concentrate on interrupt control of serial I/O within the MUART.

## Interrupt Control of Serial I/O

Recall that there are two interrupts associated with serial I/O: the Level 4 receive interrupt and Level 5 transmit interrupt. But, there are several different sources of these receive and transmit interrupts. Table 4-1 lists the possible Level 4 and Level 5 interrupt sources.

Table 4-1  
Level 4 and Level 5 Interrupt Sources

Level 4: Receive Interrupt	Level 5: Transmit Interrupt
Receive Buff. Full (RBF = 1)	Transmit Buff. Empty (TBF = 1)
Framing Error (FE = 1)	Transmit Shift. Empty (TBE = 1)
Overflow Error (OE = 1)	Break-in Detect (BD = 1)
Parity Error (PE = 1)	
Break Detect (BD = 1)	

Now, if a Level 4 receive interrupt is generated, the MPU must determine which source caused the interrupt. How do you suppose this is done? Right, the MPU reads the MUART Status Register and checks the RBF, FE, OE, PE, and BD bits to see which source caused the interrupt. The Level 4 receive interrupt service routine would then branch to the corresponding subroutine to handle the specific interrupt source.

A flowchart of a typical Level 4 receive interrupt service routine is shown in Figure 4-21. This routine assumes that the MUART has already been initialized for the receive operation. The routine first checks the BD bit in the Status Register to see if the interrupt was caused by the receipt of a break character. If the BD bit is set, a break character routine is entered. If the BD bit is cleared, the MPU checks to see if the Receive Buffer Register is full by checking the RBF bit. If the RBF bit is set, then each of the receive error status bits are checked. The respective error handling routine must be entered if any of the error bits are set, indicating an error in the received word. If no errors are found, the character is then read and stored in an area of memory referred to as the receive message buffer. Since the received characters are stored one after another in the receive message buffer, the buffer address pointer needs to be incremented before control returned to the main program. Of course, the interrupt service routine is executed for each character received.

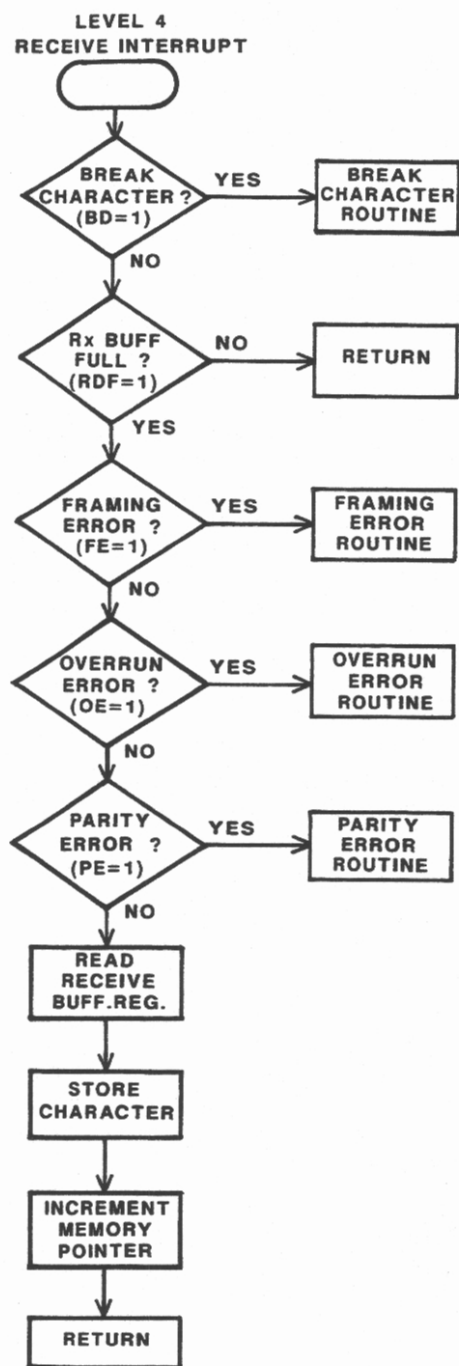


Figure 4-21

A typical receive interrupt service routine.

Likewise, if a Level 5 transmit interrupt is generated, the MPU must read the Status Register and check the TBE, TRE, and BD bits to see which source caused the transmit interrupt. Again, the service routine must be written to branch to the respective subroutine to handle the specific interrupt source.

A flowchart of a typical Level 5 transmit subroutine and interrupt service routine is shown in Figure 4-22. A transmit subroutine is called when a message must be transmitted. It is assumed that the MUART is already initialized for the transmit operation. We also must assume that the characters that make up the message to be transmitted are stored in consecutive memory locations in an area of memory called the **transmit message buffer**. In addition, we must insert a **delimiting character** after the last message character in the transmit message buffer to indicate the end of the message. This delimiting character can be any unused character for a given application.

To get the process started, the transmit subroutine must enable the transmit interrupt and write the first character to the MUART Transmit Buffer Register. When the MUART transfers the character from the Transmit Buffer Register to the Transmit Shift Register, the TBF bit in the Status Register sets and a Level 5 interrupt is generated. The Level 5 interrupt service routine shown in Figure 4-22 checks the BD and TRE bits first to see if the interrupt source is due to a break-in or the Transmit Shift Register being empty. If not, the character to be transmitted is checked to see if it is the delimiting character. If it is the delimiting character, the transmission is complete, the transmit interrupt is disabled, and the MPU returns to the main program. If it is not the delimiting character, the next character is written to the Transmit Buffer Register and the transmit message buffer pointer is incremented to point to the next character to be transmitted. The MPU then returns to the main program until another Level 5 interrupt is received, indicating that the MUART is ready for the next character.

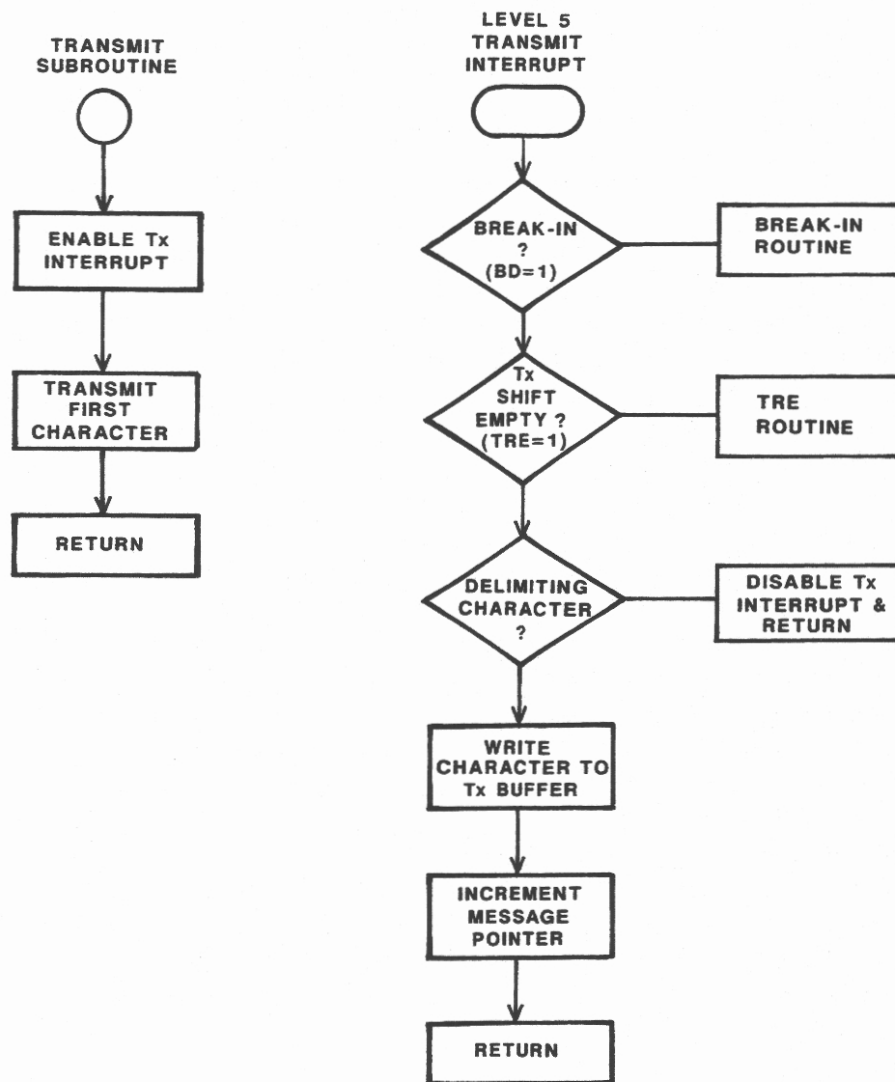


Figure 4-22

A typical transmit subroutine and associated interrupt service routine.

## Self-Test Review

30. List the user-accessible registers that are associated with the serial I/O section of the MUART.
31. Explain how the MUART registers perform a serial transmit and a serial receive operation.
32. What is a break-in, and how is it detected by the MUART?
33. Assume the MUART is decoded for port addresses 50H - 5FH. Write the 8085 code required to configure the MUART for serial I/O as follows:
  - Seven character bits.
  - Even parity.
  - Two stop bits.
  - 9600 baud for both the transmit and receive operations from the internal clock frequency.
  - Break-in feature disabled.
34. How do you perform a software versus a hardware reset with the MUART.
35. What is the functional difference between a software and a hardware reset within the MUART.
36. Can a single break character be transmitted by the MUART. If so, how?
37. What functions does Command Register 1 perform as related to serial I/O?
38. What functions does Command Register 2 perform as related to serial I/O?
39. What functions does Command Register 3 perform as related to serial I/O?
40. What functions does the Modification Register perform as related to serial I/O?
41. What status indications are provided by the MUART Status Register as related to serial I/O?



## Answers

30. Transmit Buffer Register  
Receive Buffer Register  
Command Registers 1, 2, and 3  
Modification Register  
Status Register
31. A character to be transmitted is written by the MPU to the MUART Transmit Buffer Register. The MUART then transfers the character to the Transmit Shift Register, where it is shifted out on the *TxD* line.

A character being received on the *RxD* line is shifted into the Receive Shift Register. The MUART then transfers the character to the Receive Buffer Register where it can be read by the MPU as parallel data.

32. A break-in is when the peripheral device pulls the MUART transmit line low after the last STOP bit of a character has been transmitted. This feature is generally used with half-duplex communication to allow for transmit and receive on a single line. To use the break-in feature, you must connect the *TxD* pin to the *P16* pin on the MUART. In addition, the BRKI bit in Command Register 1 must be set. The break-in condition is then detected through a Level 5 transmit interrupt with the BD bit in the Status Register set.
33. To configure the MUART as stated in Question 33, Command Register 1 must be loaded with the binary value 0110 0000, or 60H. Also, Command Register 2 must be loaded with the binary value 1111 0100, or F4H. The required 8085 code is:
- ```
MVI A,60H
OUT 50H
MVI A,F4H
OUT 51H
```
34. A software reset on the MUART is performed by setting the RST bit in Command Register 3. A hardware reset is performed by placing a logic 1 on the MUART *RESET* pin.
35. A hardware reset does the same as a software reset, but in addition clears all the Command Registers, the Mode Register, the Port 1 Control Register, and the Modification Register.
36. Yes, by setting the SBRK bit in Command Register 3.

37. Command Register 1 can be used to perform the following serial I/O functions:
  - Set the character length (L1,L0).
  - Set the number of stop bits (S1,S0).
  - Enable/disable the break-in detect feature (BRKI).
38. Command Register 2 can be used to perform the following serial I/O functions:
  - Enable/disable parity generation and checking (PEN).
  - Set even or odd parity (EP).
  - Prescale the MUART CLK input frequency (C1,C0).
  - Set the serial baud rate (B3 - B0).
39. Command Register 3 can be used to perform the following serial I/O functions:
  - Enable/disable the receiver section (RxE).
  - Transmit a single break character (SBRK).
  - Transmit a continuous break condition for any desired length (TBRK).
  - Software reset the MUART (RST).
40. The Modification Register can be used to perform the following serial I/O functions:
  - Adjust the sampling point of the received bits (RS4-RS0).
  - Enable/disable the transmission mode to detect the reception of a character transmitted by the MUART (TME).
  - Disable the start bit check (DSC).
41. The MUART Status Register provides the following status indications as related to serial I/O:
  - Receive Buffer Register full (RBF).
  - Transmit Buffer Register empty (TBE).
  - Transmit Shift Register empty (TRE).
  - Break character received or break-in detect (BD).
  - Parity error (PE).
  - Overrun error (OE).
  - Framing error or transmitted character received (FE).

## UNIT SUMMARY

The two general methods used to represent binary serial data in small microcomputer systems are mark/space and frequency modulation, sometimes called FSK. Mark/space data is usually used for serial communication between the MPU and its peripherals and is represented using TTL, CMOS, RS232C, or 20 mA signals. FSK data is generally used for MODEM communication over the telephone network and involves the use of two frequencies (1200 Hz and 2400 Hz) to represent binary data.

Two common methods used to synchronize the communication of serial data are asynchronous and synchronous serial communication. Most low-to-medium speed systems employ asynchronous communication where ASCII data must be framed using START and STOP bits. Synchronous serial communication does not require START and STOP bits, but must be synchronized via a common clock signal.

Baud rate is the total number of bits being transmitted per second, while DBPS is the number of data bits per second. Baud rate and DBPS are not the same in asynchronous systems due to the START and STOP bits.

Serial data can be channeled between devices using simplex, half-duplex, or full-duplex. Simplex is one-way communication on one line. Half-duplex is two-way communication on one line. Full-duplex is two-way communication on two lines.

Parallel data can be converted to serial data using software or hardware techniques. Software conversion with the 8085 involves the use of the 8085 *SOD* and *SID* lines. Data in the bit 7 position of the accumulator can be shifted out the *SOD* line for serial transmission, while serial data being input on the *SID* line can be shifted through the bit 7 position of the accumulator to form parallel data. The Carry bit in the MPU is crucial for both shifting operations.

Hardware conversion of asynchronous data requires a UART device which frees the MPU of the conversion task. The 8256 MUART has an internal serial I/O section that performs the UART function. The serial I/O section of the MUART employs a Transmit Buffer Register, a Receive Buffer Register, three Command Registers, a Modification Register, and a Status Register. The bits within these registers control the serial word size, baud rate, parity, and interrupt features of the serial I/O section, just to mention a few of the available control features. In addition, the Status Register bits indicate transmission/reception status, parity error, overrun error, framing error, and break conditions.

Interrupt control is normally employed to control serial I/O operations. Within the MUART, the Level 4 interrupt is the receive interrupt and the Level 5 interrupt is the transmit interrupt. The Level 4 receive interrupt, along with the Status Register bits, is used to detect when the Receive Buffer Register is full, a framing error, a parity error, an overrun error, or a break character. The Level 5 transmit interrupt, along with the Status Register bits, is used to detect when the Transmit Buffer or Transmit Shift registers are empty, or a break-in has occurred on the TxD line.



*Unit 5*

**PROGRAMMABLE TIMERS**

## CONTENTS

|                                                    |      |
|----------------------------------------------------|------|
| Introduction .....                                 | 5-3  |
| Unit Objectives .....                              | 5-4  |
| Programmable Timers .....                          | 5-5  |
| General Programmable Timer Concepts .....          | 5-6  |
| The 6840 Programmable Timer Module, or PTM .....   | 5-9  |
| I/O Diagram .....                                  | 5-9  |
| PTM Registers .....                                | 5-10 |
| Counters .....                                     | 5-10 |
| Latches .....                                      | 5-11 |
| Control Registers .....                            | 5-11 |
| Status Register .....                              | 5-16 |
| Self-Test Review .....                             | 5-19 |
| Answers .....                                      | 5-20 |
| How to Address and Initialize the 6840 PTM .....   | 5-22 |
| 6840 PTM Interfacing and Addressing .....          | 5-22 |
| PTM Initialization .....                           | 5-26 |
| Self-Test Review .....                             | 5-28 |
| Answers .....                                      | 5-29 |
| Using the PTM .....                                | 5-31 |
| Timer Output Mode Tasks .....                      | 5-31 |
| Interrupt Generation .....                         | 5-31 |
| Continuous Waveform Generation .....               | 5-34 |
| One-Shot Pulse Generation .....                    | 5-43 |
| Timer Input Mode Tasks .....                       | 5-49 |
| Period Measurement .....                           | 5-50 |
| Pulse Measurement .....                            | 5-53 |
| Self-Test Review .....                             | 5-56 |
| Answers .....                                      | 5-58 |
| The Counter/Timer Section of the MUART .....       | 5-64 |
| Counter versus Timer .....                         | 5-64 |
| Configuring the MUART Timer/Counters .....         | 5-67 |
| Counter/Timer Mode Control (Bits CT3, CT2) .....   | 5-67 |
| Timer 5 Control (Bit T5C) .....                    | 5-68 |
| Cascade Timers (Bits T35, T24) .....               | 5-68 |
| Selecting the Internal Timer Clocking Source ..... | 5-70 |
| Self-Test Review .....                             | 5-71 |
| Answers .....                                      | 5-72 |
| Unit Summary .....                                 | 5-73 |

## ***Unit 5***

# **PROGRAMMABLE TIMERS**

## **INTRODUCTION**

Many dedicated microcomputer applications require precise timing. Such applications include automobiles, appliances, traffic control, machine tool control, and industrial process control. As you are aware, software time delays can be created with the MPU. Time intervals can also be measured using software techniques. However, in performing these tasks, the MPU is not free to handle other required tasks. The result is inefficiency.

Fortunately, there are single-chip LSI devices available that are programmable and perform a variety of timing tasks. These devices are called **programmable timers**. Once initialized for a given timing task, they do not require service from the MPU until the timed interval is completed. Programmable timers are used to create time delays, generate continuous and one-shot pulses, measure input signal periods and pulse widths, and measure time between external events. Time intervals of just a few microseconds or many years can be generated or measured using programmable timers.

In this unit, you will learn how to use a programmable timer with a microprocessor system to perform a variety of timing tasks. In addition, you will learn about the internal timer capabilities of the 8256 MUART.



## UNIT OBJECTIVES

1. Explain the general function and structure of a programmable timer.
2. Describe the internal structure and register functions of a Programmable Timer Module (PTM).
3. Explain the function of the PTM I/O lines.
4. Interface the PTM to a microprocessor-based system.
5. Address and initialize the PTM.
6. Generate time delays, continuous waveforms, and one-shot pulses using the PTM.
7. Measure the period and/or pulse width of an externally applied signal using the PTM.
8. Measure the time between two external events using the PTM.
9. Initialize the PTM to perform the above mentioned timing tasks.
10. Explain how to use the internal 8256 MUART timer.

## PROGRAMMABLE TIMERS

A programmable peripheral control device that can perform external timing operations is called a programmable timer. You can program these devices with the MPU to perform the following tasks:

- Provide time delays.
- Generate control signals.
- Generate continuous and one-shot square wave pulses.
- Measure the time between external events.
- Measure the period of an external waveform.
- Measure the duration of an external pulse.
- Provide a real-time clock.
- Provide external motor control.

The main purpose of a programmable timer is to allow the MPU to be free of the timing task. Once the timer is initialized by the MPU, additional service is not required by the MPU until the timer generates an interrupt. Thus, the timer provides the time delay task, and not the MPU. During the timed intervals, the MPU is free to perform other tasks. A programmable timer is especially valuable in process control applications, where various control tasks must be entered at specific intervals.

In this section, you will learn about programmable timers in general. Then, you will be introduced to the 6840 programmable timer module, or PTM, to prepare you for the next section.

## General Programmable Timer Concepts

Most programmable timers contain multiple timer sections. These timer sections can operate independently or be cascaded together to provide longer timed intervals. Each timer section contains a counter register and a control register as shown in Figure 5-1. A status register, which is common to all the timer sections, is also sometimes included.

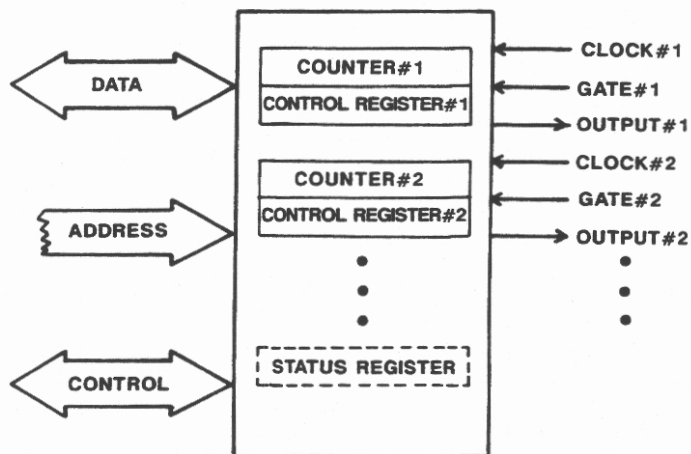


Figure 5-1  
A typical programmable timer.

The counter registers are usually 16-bit down counters. When preset with a value, the counter will count down at a rate that is derived from a timer clock signal. When the counter reaches zero, an interrupt is generated to the MPU, or external logic is generated via the timer output line.

The term time-out, or  $T_O$ , is used to indicate when a counter reaches zero, or is stopped. By providing an output logic transition each time a time-out point is reached, the timer can generate continuous and one-shot square wave pulses. We will define a timer section to be in its output mode whenever it generates output logic as a result of the internal counter activities. Several output mode tasks are illustrated in Figure 5-2.

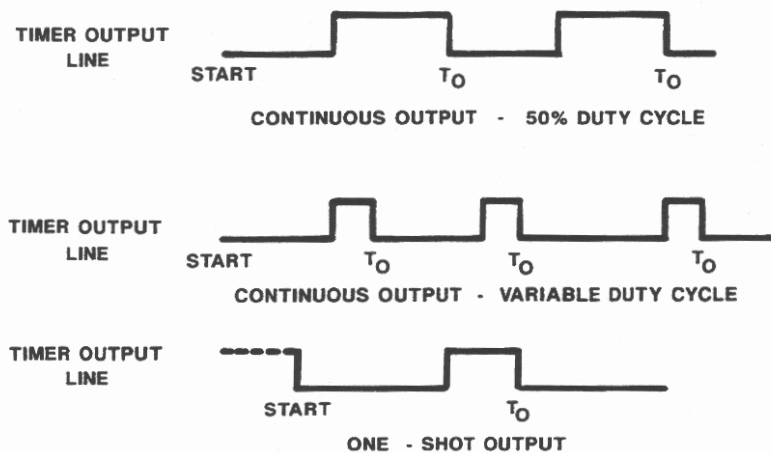


Figure 5-2  
Programmable timer output tasks.

The timer counter can also be made to start and stop by the application of an external signal on its *gate input* line. An externally applied signal transition in one direction will start the timer counter. Then, a subsequent signal transition in the same or opposite direction is used to stop the counter. By reading the counter's contents before and after the count, you can determine the time between external events, length of a pulse, or period of an externally applied waveform. We will define a timer section to be in its input mode whenever its counter is controlled by external logic being applied to its gate input line. Several input mode tasks are illustrated in Figure 5-3.

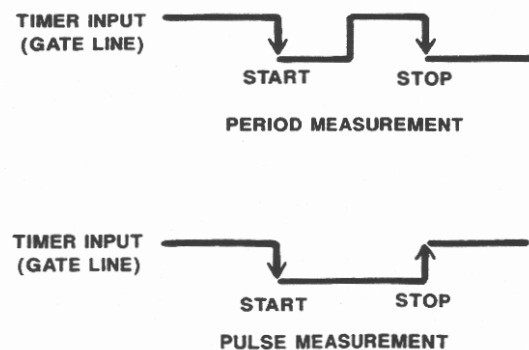


Figure 5-3  
Programmable timer input tasks.

Each timer control register is used to determine the function of its respective counter. You will configure the control registers to perform one of the various timing tasks illustrated in Figures 5-2 and 5-3 by writing a value to the control register. Configuring the timer control registers is referred to as **timer initialization**.

A status register is included in some programmable timers to indicate the interrupt status of the different timer sections. When this is the case, only one interrupt line is used to generate an interrupt to the MPU. You can determine which timer caused the interrupt to be generated by reading the status register. Recall that this technique was also used in the ACIA to delineate between a transmit and a receive interrupt.

In addition to the internal registers shown in Figure 5-1, each timer section has an associated clock line, gate line, and output line. Each of these lines is represented by an external pin on the device. The clock line can be used to establish the respective timer count rate with an externally applied clock signal. The gate line is used to apply external counter control signals to the respective timer section for the input mode of operation. Finally, the output line is used during the output mode of operation. It supplies output logic as the result of internal counter activities.

Most microprocessor families include a programmable timer. In the Intel 8080/8085 family, two programmable timing devices of particular interest are the 8253 Programmable Interval Timer and the 8256 MUART. In the Motorola 6800 family, the programmable timing device is the 6840 Programmable Timer Module, or PTM. In addition, many of the newer single-chip microcomputers (such as Intel's 8048, 8049, and Motorola's 6801, 6805, 68HC11) contain internal programmable timers.

You will now become acquainted with the Motorola 6840 PTM. Then you will learn how the 8256 MUART incorporates an internal "on-chip" programmable timer section.

We will discuss both the 6840 PTM and the 8256 MUART timer section as a comprehensive introduction to programmable timers.

## The 6840 Programmable Timer Module, or PTM

As mentioned earlier, the programmable timing device in the Motorola 6800 family is the 6840 PTM. This device was designed to be directly compatible with 6800 family processors like the 6800, 6802, and 6809; however, it can also be used with other family processors such as the Intel 8080, 8085, and 8088. Using the 6840 PTM, you can perform all the input and output mode timing tasks, discussed earlier in this section.

### I/O Diagram

The 6840 PTM is a 28-pin NMOS device. The I/O diagram of the PTM in Figure 5-4 shows its input and output lines. Data is transferred to and from the PTM, a byte at a time, by the data bus lines *D0* through *D7*. The PTM monitors five or more of the MPU's address lines via its two chip select, or *CS* lines, and three register select, or *RS* lines. The chip select lines enable the PTM and must be activated by partial or full decoding of the MPU address bus lines. The register select lines are used to access the internal registers of the PTM.

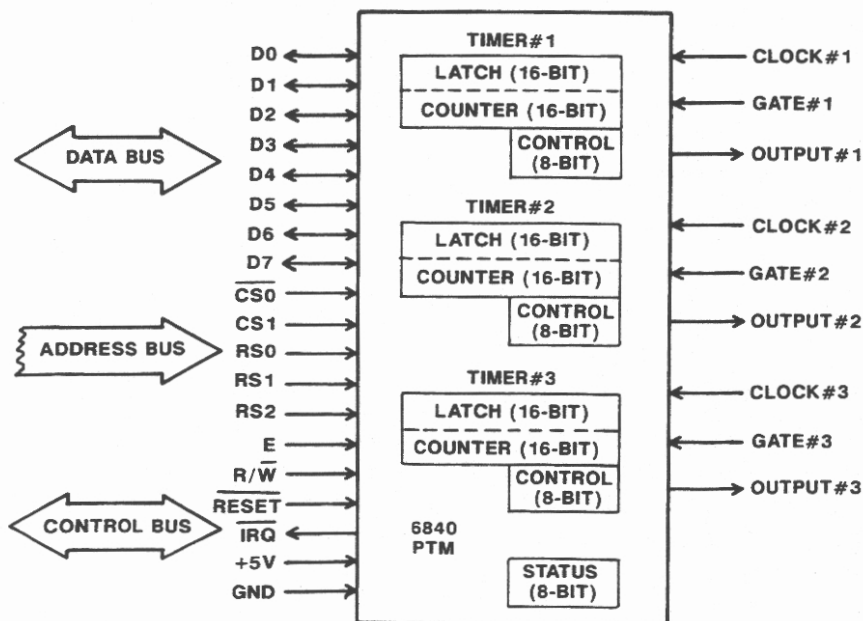


Figure 5-4

Internal structure and I/O diagram of the 6840 programmable timer module, or PTM.

The PTM also connects to several of the MPU's control lines. The PTM  $R/\overline{W}$  line is connected directly to the MPU  $R/\overline{W}$  line. This line determines the direction of data transfer between the MPU and PTM.

The  $E$  line of the PTM is connected directly to the MPU clock line. This line provides the basic timing signal for data transfers between the MPU and PTM.

The PTM has one interrupt request line labeled  $\overline{IRQ}$ . This line allows the PTM to get the attention of the MPU when one of its internal timers has timed-out. The PTM  $\overline{IRQ}$  line can be connected directly to any of the MPU interrupt lines.

The PTM  $\overline{RESET}$  line is usually connected directly to the MPU  $\overline{RESET}$  line. This allows the PTM registers to be reset to a known condition at the same time the MPU is reset.

Each timer section of the PTM has an associated clock, gate, and output line. As mentioned earlier, the clock line is an input line used to establish the respective timer count rate; the gate line is an input line used to apply external counter control signals to the respective timer section; and the output line is used to supply output logic as the result of internal timer activities.

## PTM REGISTERS

The PTM contains three independent timer sections and a common status register as shown in Figure 5-4. Each of the three timer sections contain an addressable 16-bit counter, 16-bit latch, and an 8-bit control register. The timer sections can be operated independently or cascaded together to provide for longer timed intervals.

## Counters

The timer counters are 16-bit down counters and are *read-only* registers. They are used to provide precisely timed intervals as a result of their beginning count value and their clocking rates. The counters can be read by the MPU "on the fly" or when in a hold state due to a stop signal present on the respective gate input line. The count rate of each counter can be established independently by its respective clock input line, or by the internal PTM clock signal.

## Latches

The timer latches are 16-bit *write-only* storage registers which are used to load the counters with a beginning count value.

## Control Registers

The control registers are 8-bit *write-only* registers that are used to control the operation of each respective timer section. The key to understanding the operation of the PTM is understanding the various control register bit functions.

The bit structure of the control registers is shown in Figure 5-5. You will note that bits 1 through 7 of each control register perform the same function. However, bit 0 of each control register performs a unique function. In this section, we intend to provide you with a general understanding of the functions that the timer control register bits perform. Then, in the next section, you will see how the control registers are used to initialize the PTM and select its various control options.

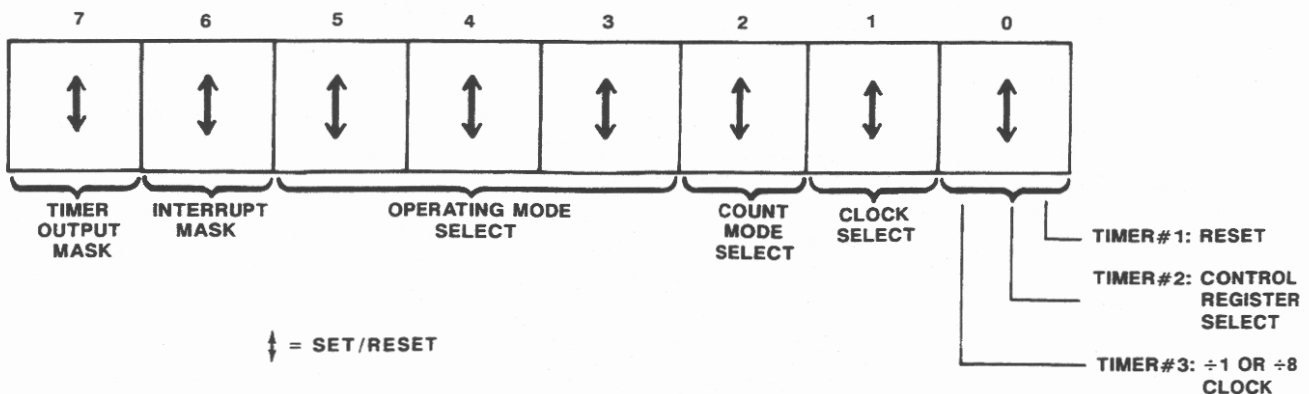


Figure 5-5  
PTM control register bit functions.



The least significant bit, or bit 0, of each timer control register performs a unique function. This bit in the timer #1 control register is used to provide a software reset for the PTM. By setting bit 0 of the timer #1 control register, a reset will occur and accomplish the following:

- All of the PTM timer operations are stopped.
- All interrupt flags in the status register are cleared.
- All timer latches retain the data last written to them.
- All control register bits remain unchanged.

In the #2 control register, bit 0 is used to select between the timer #1 control register and timer #3 control register when addressing the PTM. When cleared, the timer #3 control register is selected; when set, the timer #1 control register is selected. This bit is used in the PTM initialization process to select between the timer #1 and timer #3 control registers.

In the timer #3 control register, bit 0 provides a clock divide option for timer #3. When set, the timer #3 clock is divided by eight, or  $\div 8$ . Thus, a slower count rate can be achieved. When cleared, timer #3 is clocked directly, or  $\div 1$ , from its clocking source.

A summary of the function of control register bit 0 for each timer section is provided in Figure 5-6.

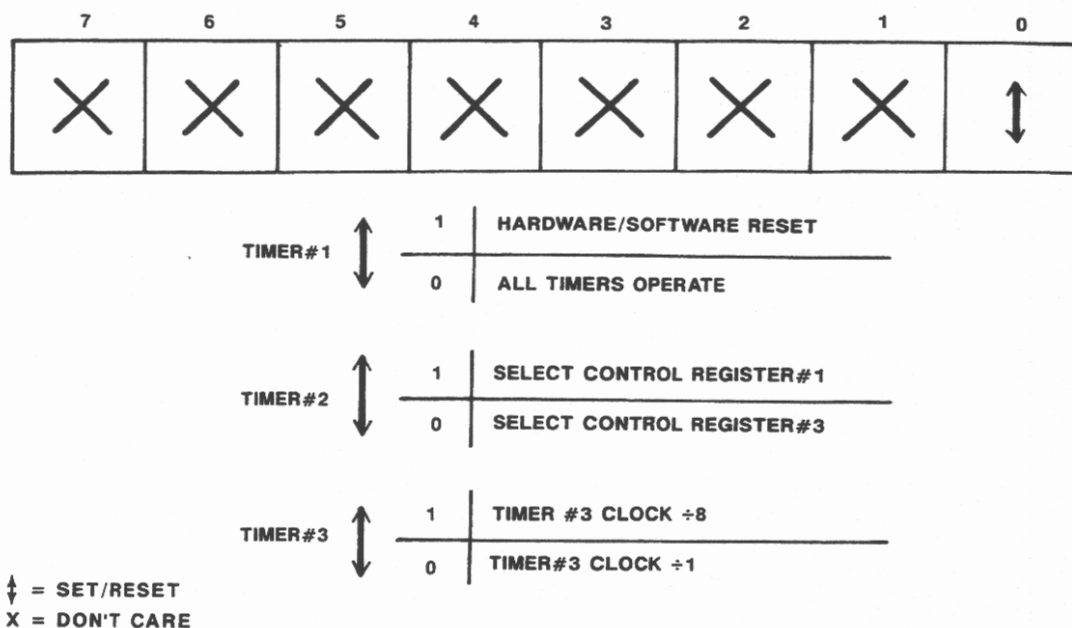


Figure 5-6  
Control register bit 0 summary.

The next bit in each control register, **bit 1**, is used to select between an internal or external timer clocking source. Recall that each timer section has an external clock input line. When bit 1 of the respective control register is cleared, the timer counter is clocked from its externally applied clock line signal. When bit 1 is set, the timer counter is clocked from the internal PTM clock. The internal clock is provided by the *E* line of the PTM, which is normally connected directly to the MPU clock line. Thus, with a typical 1 MHz *E* clock rate, the internal timer counter will count at 1 MHz rate when internal clocking is selected. The function of bit 1 is summarized in Figure 5-7.

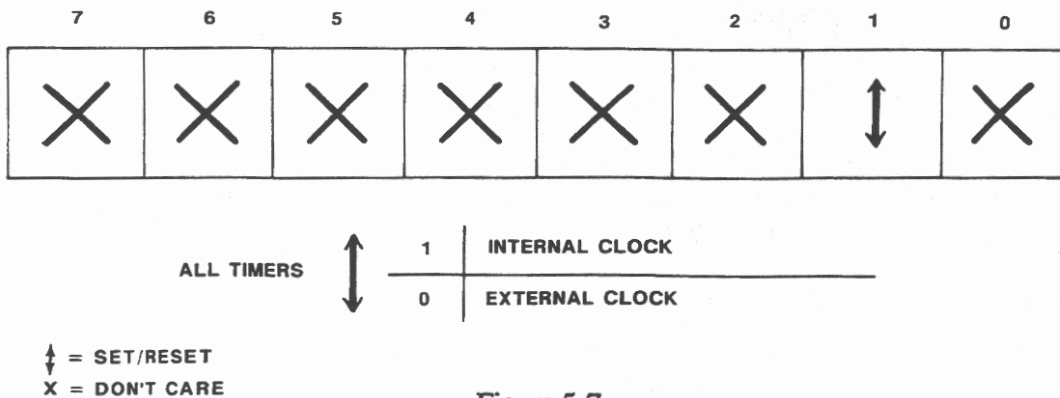


Figure 5-7

Control register bit 1 summary.

The function of **bit 2** of the timer control registers is to select between a normal *16-bit count* and a *dual 8-bit count* during any one of the timer output operating modes. Recall that the two possible timer output modes are continuous and one-shot pulse generation. A normal *16-bit count* is used to generate a continuous waveform with a 50% duty cycle. However, a dual 8-bit count is used to generate a waveform with duty cycles other than 50%. When bit 2 of the control register is cleared, the 16-bit count is selected. When set, the dual 8-bit count is selected. The use of this bit to obtain different duty cycle outputs will be demonstrated in the PTM experiments. The function of bit 2 is summarized in Figure 5-8.

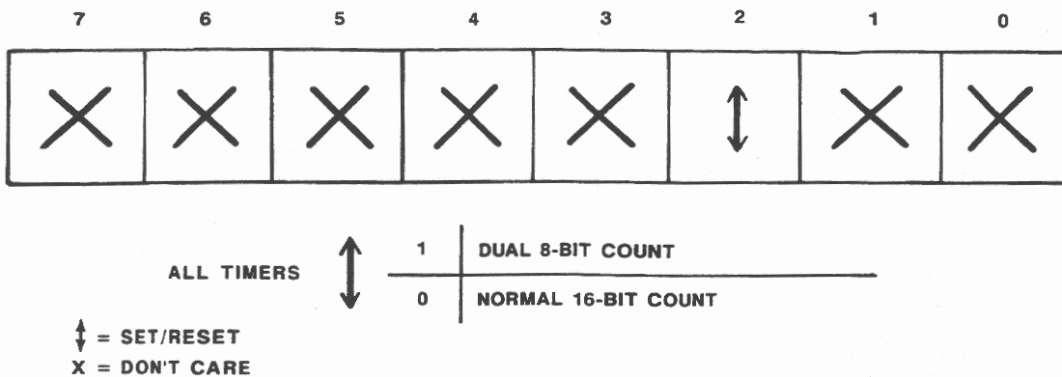


Figure 5-8

Control register bit 2 summary.

Bits 3, 4, and 5 of each timer control register are used collectively to select the operating mode of the respective timer section. Recall from the general discussion on timers that we defined two general operating modes: the timer *input mode* and timer *output mode*. The two tasks that can be performed with the timer input mode are *period and pulse measurement*. With the timer output mode, the two tasks that can be performed are *continuous and one-shot pulse generation*. The function of bits 3, 4, and 5 of the control register is to select one of these four input/output operating tasks. The selection criteria is provided in Figure 5-9. The don't-cares, or X's, in Figure 5-9 provide various features within the selected task and will be discussed in the next section.

| OPERATING MODE SELECT FIELD |   |   |                          |
|-----------------------------|---|---|--------------------------|
| 5                           | 4 | 3 | TASK SELECTED            |
| 0                           | X | 0 | CONTINUOUS WAVEFORM GEN. |
| 1                           | X | 0 | ONE-SHOT PULSE GEN.      |
| X                           | 0 | 1 | PERIOD MEAS.             |
| X                           | 1 | 1 | PULSE MEAS.              |
|                             |   |   | MODE                     |
|                             |   |   | OUTPUT                   |
|                             |   |   | INPUT                    |

Figure 5-9

Control register bits 3, 4, and 5 summary.

The function of bit 6 is to enable and disable the timer interrupts. Recall that an interrupt may be generated when a timer times-out, or is stopped. If bit 6 of the respective timer control register is set, the interrupt is enabled and will be generated via the PTM's  $\overline{IRQ}$  line when the respective timer times-out or is stopped. However, if bit 6 is cleared, the interrupt is not generated, it is disabled. The function of bit 6 is summarized in Figure 5-10.

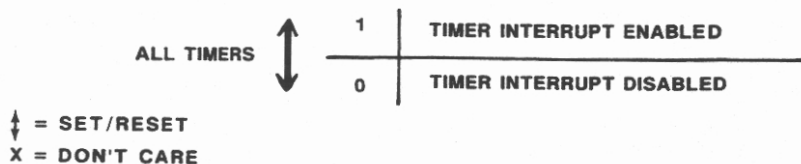
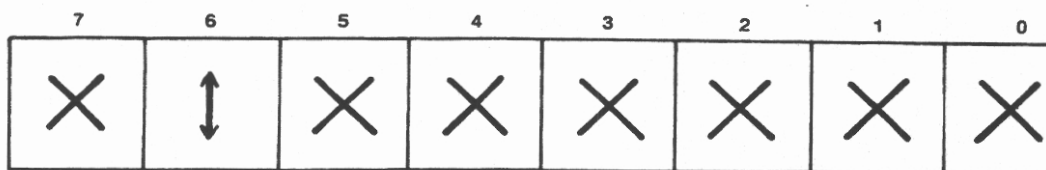


Figure 5-10

Control register bit 6 summary.

Finally, the function of bit 7 of each control register is to enable and disable the output line of the respective timer section. When bit 7 is set, the timer output line is enabled. When cleared, the output line is disabled. The function of bit 7 is summarized in Figure 5-11.

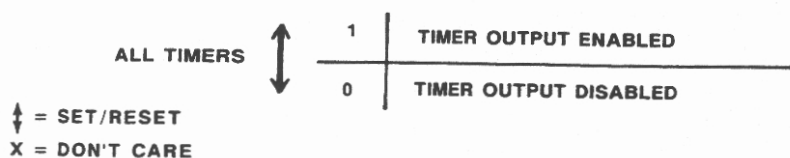
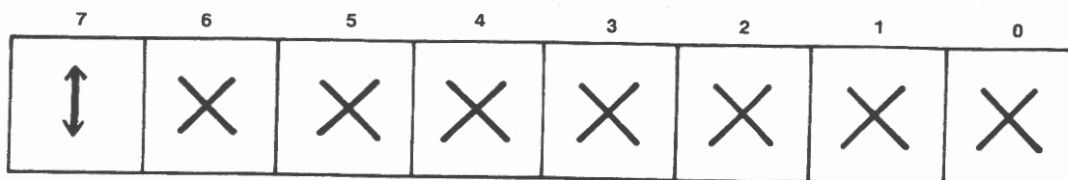


Figure 5-11  
Control register bit 7 summary.

Now, let's look at a sample PTM configuration. Suppose the control register for timer #1 contains the hex value 92H as shown in Figure 5-12. This configuration will control timer #1 as follows:

- Bit 0 cleared means that all timers are in the operating mode and no software reset is asserted.
- Bit 1 set selects internal clocking for timer #1.
- Bit 2 cleared provides for normal 16-bit counting.
- The value of the mode select field (bits 5, 4, and 3) is 010, which selects the continuous pulse generation mode.
- Bit 6 cleared disables a timer #1 interrupt from being generated on the PTM  $\overline{IRQ}$  line.
- Bit 7 set enables the timer #1 output line for continuous pulse generation.

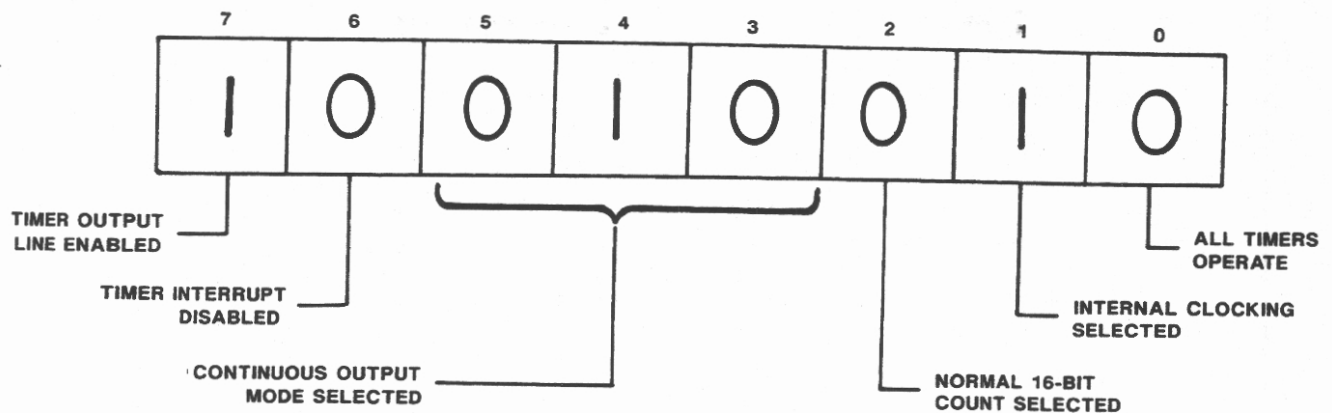


Figure 5-12

Timer #1 control register configured for continuous output with internal clocking.

## Status Register

The sole purpose of the PTM status register is to indicate interrupt status of the three timer sections. The status register is a read only register that contains four interrupt status flags: one flag for each of the three PTM timers, and a composite interrupt flag as shown in Figure 5-13. You will note that bits 3 through 6 are not used and permanently set to zeros.

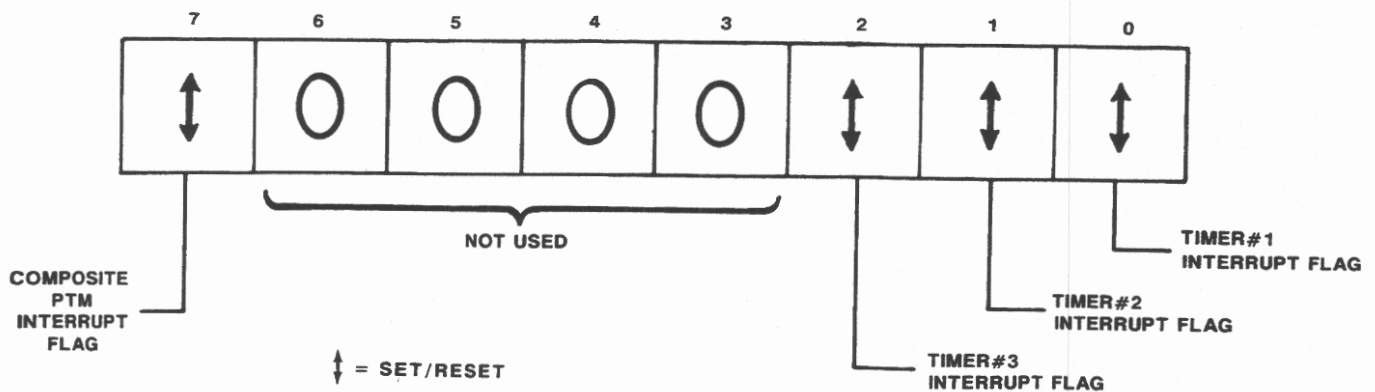


Figure 5-13

PTM status register bit functions.

Recall that when a timer times-out, an interrupt can be generated. In the case of the 6840 PTM, the respective timer interrupt flag in the status register always sets when a time-out occurs. However, an interrupt is not generated unless enabled by bit 6 of the respective timer control register.

The relationship between the individual timer interrupt status flags and the composite interrupt flag is illustrated in Figure 5-14. The composite flag will set when any of the individual timer flags set *and* the timer's interrupt is enabled by bit 6 of its control register. When the composite flag sets, an interrupt is generated via the PTM  $\overline{IRQ}$  line. Note from the functional diagram in Figure 5-14 that bit 6 of a given timer control register will close the switch to enable a timer interrupt, and open the switch to disable the interrupt.

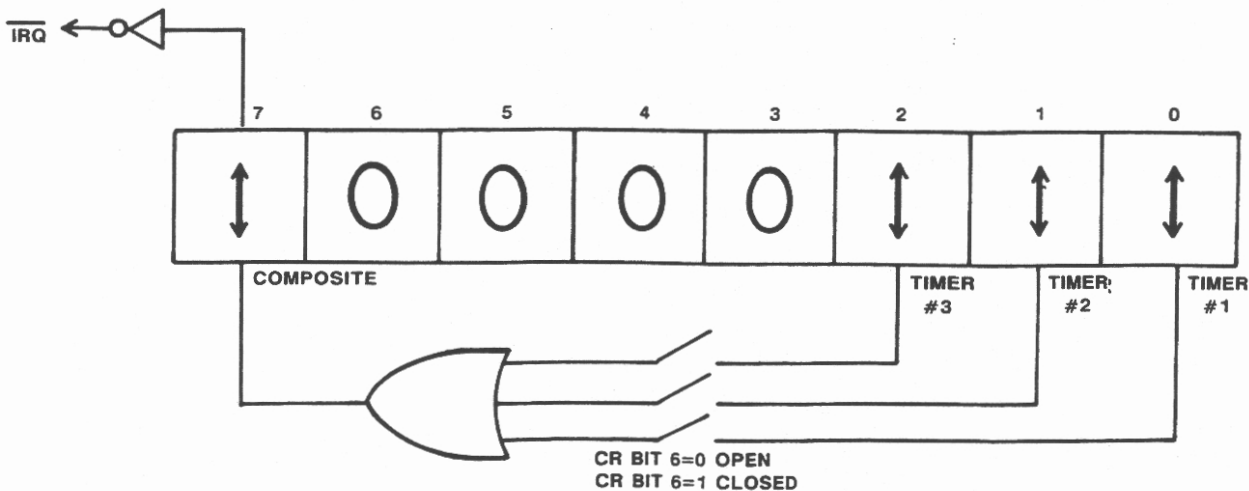


Figure 5-14

Relationship between the individual timer interrupt flags and the composite PTM interrupt flag.

Once an interrupt is generated by the PTM  $\overline{IRQ}$  line, you must determine which timer generated the interrupt. You can do this by reading the status register and checking the individual timer interrupt status flags.

In general, the interrupt status flags are cleared by a hardware or software reset. Furthermore, the composite interrupt flag and a given individual timer flag are cleared by reading the status register, then reading the timer counter that caused the interrupt.

A summary of the control and status register bit functions is provided in Figures 5-15 and 5-16 respectively. It would be helpful to study these figures in preparation for the next section, and the PTM experiment.

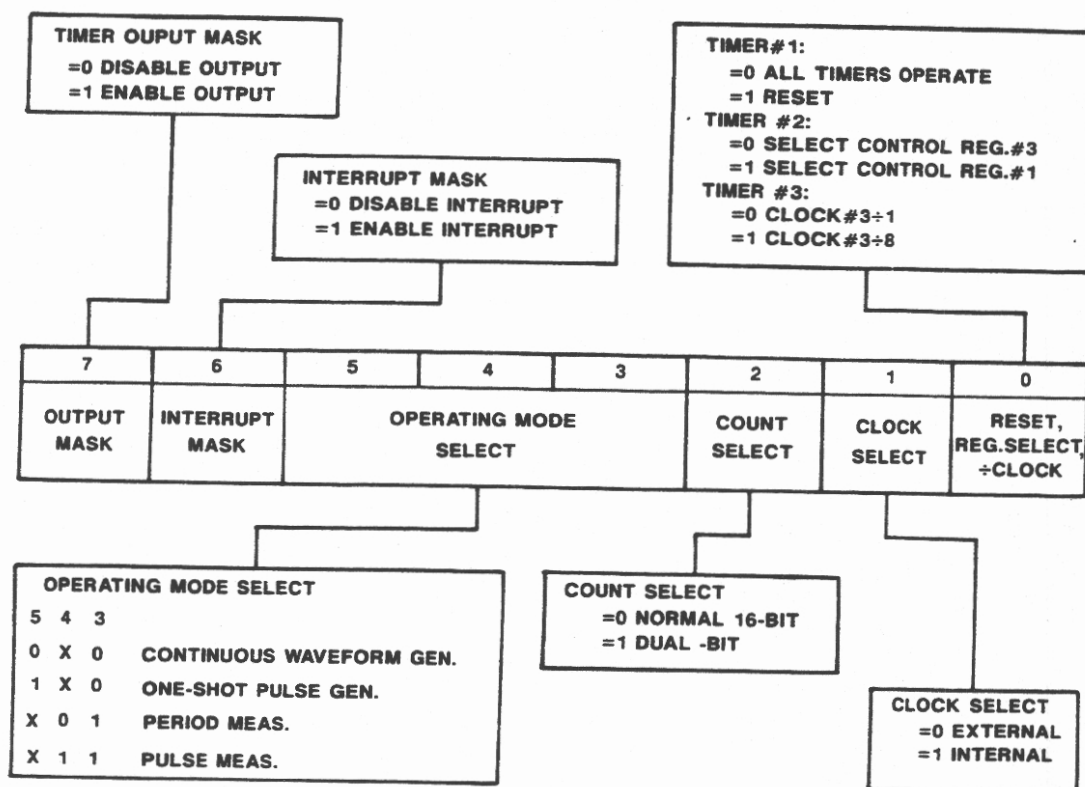


Figure 5-15  
PTM control register summary.

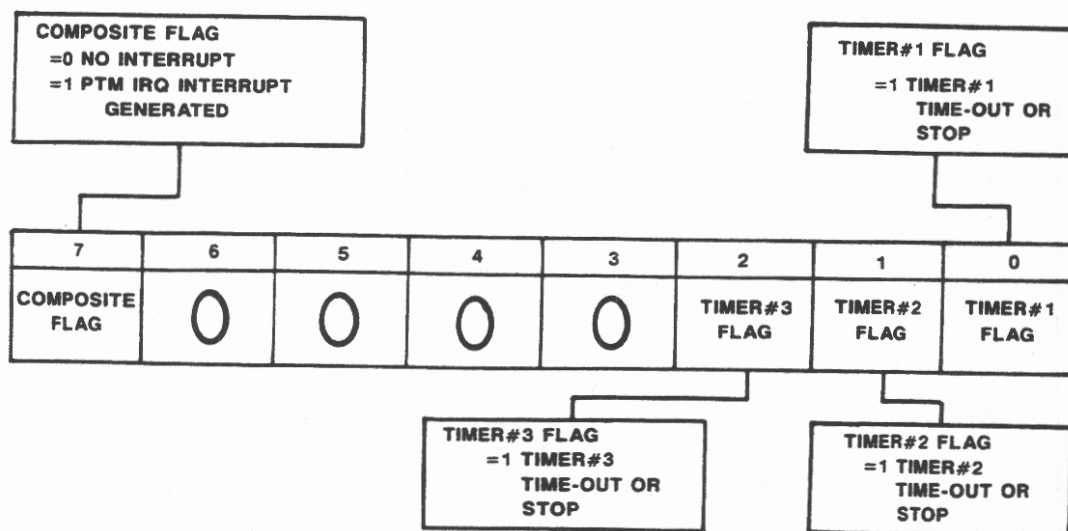


Figure 5-16  
PTM status register summary.

## Self-Test Review

1. What is the main purpose of a programmable timer? \_\_\_\_\_  
\_\_\_\_\_
2. What does the term "time-out," or TO, mean? \_\_\_\_\_  
\_\_\_\_\_
3. Describe the input and output programmable timer modes. \_\_\_\_\_  
\_\_\_\_\_
4. List two input mode tasks and two output mode tasks that are commonly performed by programmable timers. \_\_\_\_\_  
\_\_\_\_\_
5. Describe the function of the timer clock line, gate line, and output line. \_\_\_\_\_  
\_\_\_\_\_
6. What are the three registers that make up a timer section in the 6840 PTM? \_\_\_\_\_  
\_\_\_\_\_
7. Briefly describe the function of each of the three registers that make up a PTM timer section. \_\_\_\_\_  
\_\_\_\_\_
8. Which of the 6840 PTM registers is common to all three internal timer sections? \_\_\_\_\_  
\_\_\_\_\_
9. What bit(s) in the control registers have a unique function for each of the three 6840 PTM timer sections? \_\_\_\_\_  
\_\_\_\_\_
10. Suppose that the timer #3 control register contains the hex value  $4A_{16}$ . How does this control the timer? \_\_\_\_\_  
\_\_\_\_\_
11. What is the main purpose of the 6840 PTM status register? \_\_\_\_\_  
\_\_\_\_\_



## Answers

1. The main purpose of a programmable timer is to free the MPU of the timing task.
2. The term time-out, or TO, is used to indicate when a timer counter reaches zero, or is stopped.
3. When in the input mode, a timer section is controlled by external logic applied to its gate line.  
When in the output mode, a timer section generates output logic as a result of the internal counter activities.
4. Input mode tasks include period measurement, interval measurement, and pulse measurement.  
Output mode tasks include interrupt generation, continuous waveform generation, and one-shot pulse generation.
5. The timer clock line can be used to establish the timer count rate with an externally applied clock signal.  
The gate line is used to apply external counter control signals to the timer.  
The output line is used to supply output logic as a result of internal counter activities.
6. The three registers that make up a timer section in the 6840 PTM are the timer counter, timer latch, and timer control register.
7. The timer counter is a 16-bit down counter. It is a read-only register which provides precisely timed intervals as a function of its beginning count value and its clockrate.  
The timer latches are 16-bit write only storage registers which are used to load the counters with a beginning count value.  
The timer control registers are 8-bit write only registers that are used to control the operation of each respective timer section.
8. The 6840 PTM status register is common to all three internal timer sections.
9. Bit 0 in the control registers has a unique function for each of the three 6840 PTM timer sections.

10. The hex value  $4A_{16}$  stored to the timer #3 control register would control the timer as follows:
  - Bit 0 cleared selects the  $\div 1$  clock option for timer #3.
  - Bit 1 set selects internal clocking for timer #3.
  - Bit 2 cleared provides for normal 16-bit counting.
  - The value in the mode select field (bits 5, 4, and 3) is 001, which selects the period measurement input mode.
  - Bit 6 set enables the timer #3 interrupt.
  - Bit 7 cleared disables the timer #3 output line.
11. The main purpose of the 6840 PTM status register is to indicate composite and individual interrupt status of the three timer sections.

## HOW TO ADDRESS AND INITIALIZE THE 6840 PTM

In the last section, you were introduced to programmable timers in general, and the 6840 PTM in particular. In addition, you were acquainted with the I/O lines of the 6840 PTM and its internal structure.

In this section, you will learn how to interface the 6840 PTM to a microprocessor-based system. To operate the 6840 PTM, it must first be initialized to configure its control registers. The control registers must be configured for one of several operating modes. You should pay particular attention to the PTM initialization procedure discussed in this section. This will prepare you to better understand how the PTM operates and how it is used to perform various timing tasks.

### 6840 PTM Interfacing and Addressing

First, you must access the PTM by activating its two chip select lines:  $\overline{CS0}$ , and  $CS1$ . These two lines are used to enable the PTM. In order for the PTM to be enabled,  $\overline{CS0}$  must be low while  $CS1$  must be high. Since there are only two chip select lines, they are normally connected through a decoding circuit as shown in Figure 5-17. The decoding circuit in Figure 5-17 decodes the address bus for port addresses 50H through 5FH. Thus, any port address within this range will cause a low level to be applied to  $\overline{CS0}$ . As you will see shortly, the PTM only requires eight address assignments.

You will also note from Figure 5-17 that the PTM data lines ( $D0-D7$ ), read/write line ( $R/\overline{W}$ ), and  $\overline{RESET}$  line connect directly to the corresponding MPU lines. In addition, the PTM enable line ( $E$ ) connects directly to the MPU clock line for timing purposes, and the PTM interrupt line ( $\overline{IRQ}$ ) can be connected directly to any of the MPU interrupt lines.

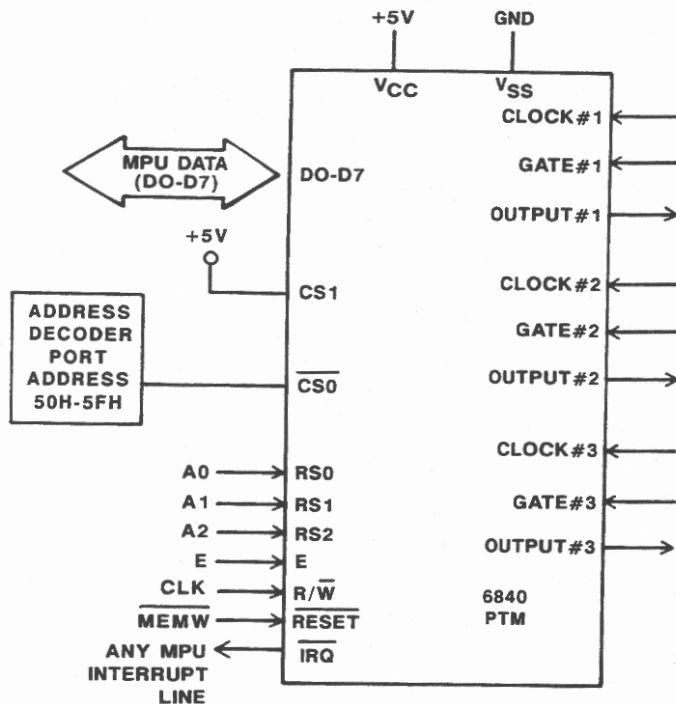


Figure 5-17

Connecting the 6840 PTM to the MPU.

Once the PTM is selected with the proper chip select inputs, you must access one of the ten registers within the PTM. The three PTM register select lines labeled  $RS0$ ,  $RS1$ , and  $RS2$  in Figure 5-17, and the  $R/\overline{W}$  line are used for the register selection process. Recall that the timer control registers and latches are *write only* registers. However, the timer counters and status register are *read only* registers. Therefore, the PTM  $R/\overline{W}$  line can be used to distinguish between these two read/write register categories during the register selection process. The timer counters or status register are accessed during a read operation, when the  $R/\overline{W}$  line is high. The timer control registers or latches are accessed during a write operation, when the  $R/\overline{W}$  line is low. Once the  $R/\overline{W}$  line determines the read/write register category, the exact register that is selected is determined by the three register select, or  $RS$ , lines of the PTM. Furthermore, bit 0 of the timer #2 control register is used during a write operation to select between the timer #1 and timer #3 control registers. The PTM register selection process is illustrated in Figures 5-18 and 5-19. A PTM write operation is illustrated in Figure 5-18, and a PTM read operation in Figure 5-19.

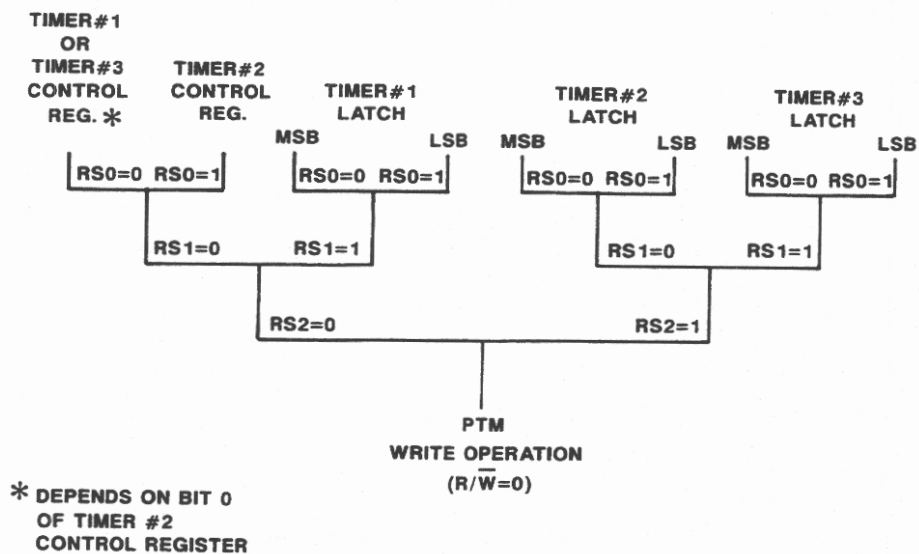


Figure 5-18  
PTM register selection for a write operation ( $R/\overline{W} = 0$ ).

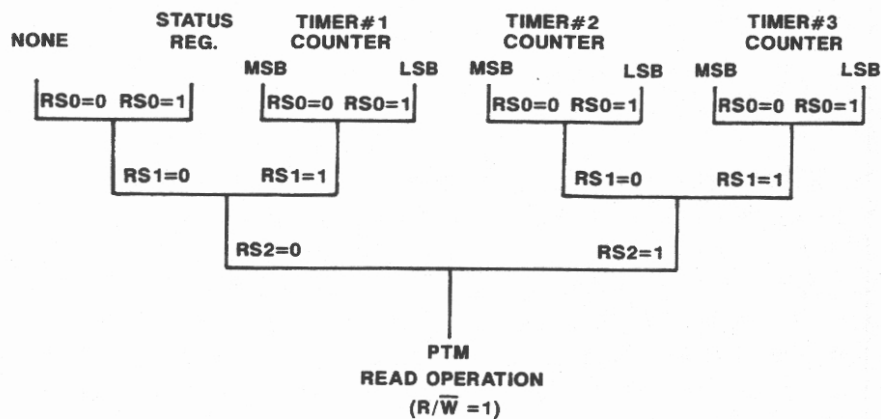


Figure 5-19  
PTM register selection for a read operation ( $R/\overline{W} = 1$ ).

You will normally connect the three PTM register select lines, *RS0*, *RS1*, and *RS2* to the three least significant MPU address lines, *A0*, *A1*, and *A2* respectively. When this is done, the interface circuit in Figure 5-17 will access the PTM registers at port addresses 50H through 57H as shown in Figure 5-20. Of course, the decoding circuit can be changed to map the PTM into a different address space. However, by connecting *RS0*, *RS1*, and *RS2* to *A0*, *A1*, and *A2* respectively, the first eight addresses of the decoded space will always provide the register selection order shown in Figure 5-20. In fact, we will assume the PTM register address map given in Figure 5-20 for all subsequent PTM discussions. In addition, this register map and the interface circuit given in Figure 5-17 are used for the PTM experiments.

As stated earlier, the timer counters and timer latches are 16-bit registers. This is why they each occupy two addresses in Figure 5-20.

| PORT ADDRESS | PTM REGISTER SELECTED WITH READ OPERATION | PTM REGISTER SELECTED WITH WRITE OPERATION |
|--------------|-------------------------------------------|--------------------------------------------|
| 50H          | NONE                                      | CONTROL REG. #1 OR #3 *                    |
| 51H          | STATUS REG.                               | CONTROL REG. #2                            |
| 52H          | TIMER #1                                  | TIMER #1                                   |
| 53H          | COUNTER                                   | LATCH                                      |
| 54H          | TIMER #2                                  | TIMER #2                                   |
| 55H          | COUNTER                                   | LATCH                                      |
| 56H          | TIMER #3                                  | TIMER #3                                   |
| 57H          | COUNTER                                   | LATCH                                      |

\* DEPENDS ON CONTROL REG. #2, BIT 0

Figure 5-20

Address versus PTM register selection for the MPU interface circuit in Figure 5-17.

## PTM Initialization

The PTM initialization procedure requires two major steps:

1. Write a beginning count value to the timer latches.
2. Write a control word to the timer control registers.

These operations must be performed sequentially in the order given.

The timer latches will normally be loaded using a 16-bit store instruction such as STX or STS. The value that is written to a given timer latch will be the beginning count value for that particular timer when the timer is started. If a value is not written to a given timer latch, the beginning count value will automatically be set to  $FFFF_{16}$  for that timer.

The control registers must be configured in the following order:

- Write a control word to control register #3.
- Write a control word to control register #2.
- Write a control word to control register #1.

The PTM control registers may be configured using any of the MPU 8-bit store instructions such as STA or OUT.

The PTM reset line is connected directly to the MPU reset line. After the system is reset, all control register bits are cleared except for bit 0 of control register #1. Thus, the reset operation accomplishes two control tasks. First, bit 0 of control register #2 is cleared and, therefore, control register #3 is accessed at address 50H, and *not* control register 1. Second, bit 0 of control register #1 is set to put all timers in a hold state until the PTM initialization procedure has been completed.

Now, recall that bit 0 of control register #2 is used to select between control register #1 and control register #3. Since a reset operation clears this bit, control register #3 is selected. Therefore, you must first write a control word to control register #3 at address 50H. Then, you must write a control word to control register #2 at address 51H which sets bit 0 to subsequently select control register #1. After bit 0 of control register #2 is set, control register #1 is accessed at address 50H. The next operation is to write a control word to control register #1 at address 50H. Thus, the required control register configuration sequence given above is completed.

Recall that bit 0 of control register #1 puts all the PTM timers in a hold state until bit 0 cleared. Configuring control register #1 is the last step in the initialization process. When writing a control word to control register #1, you will always clear bit 0 to start the timers.

A sample PTM initialization routine is provided in Figure 5-21. This routine will initialize the PTM for the continuous waveform generation task using timer #3. The program assumes that a reset has occurred prior to initialization. The beginning count value, or D755H, is stored to the timer #3 latch using the index register. The control register for timer #3 is then configured to provide the continuous waveform generation task. To start the output waveform generation, you must clear bit 0 of control register #1. Thus, bit 0 of control register #2 is first set to access control register #1, and then bit 0 of control register #1 is cleared to start the continuous waveform generation.

| SOURCE CODE                                  | COMMENTS                                                          |
|----------------------------------------------|-------------------------------------------------------------------|
| MVI A,D7H<br>OUT 56H<br>MVI A,55H<br>OUT 57H | STORE BEGINNING COUNT VALUE,<br>D755H, TO TIMER #3 LATCH          |
| MVI A,92H<br>OUT 50H                         | CONFIGURE CONTROL REGISTER #3<br>FOR THE CONTINUOUS OUTPUT TASK   |
| MVI A,01H<br>OUT 51H                         | SET BIT 0 OF CONTROL REGISTER #2<br>TO SELECT CONTROL REGISTER #1 |
| MVI A,00H<br>OUT 50H                         | CLEAR BIT OF CONTROL REGISTER #1<br>TO START THE TIMER OPERATION  |

Figure 5-21

Sample PTM initialization program for the  
MPU interface circuit in Figure 5-17.

Even though only one of the PTM timer sections is being used in this example, all of the control registers must be accessed to provide the required PTM initialization sequence.



## Self-Test Review

12. Explain how the 6840 PTM is interfaced to an MPU. \_\_\_\_\_  
\_\_\_\_\_
13. When the PTM  $R/\overline{W}$  line is high, what register(s) are selected? \_\_\_\_\_  
\_\_\_\_\_
14. Once the PTM  $R/\overline{W}$  line determines the read/write register category, what determines the exact internal PTM register that is selected? \_\_\_\_\_  
\_\_\_\_\_
15. Suppose the PTM is assigned to port addresses 80H through 87H. Construct a register address map table for the internal PTM registers.
16. What two major steps are required to initialize the PTM? \_\_\_\_\_  
\_\_\_\_\_
17. What instructions are normally used to write a beginning count value to the timer latches? \_\_\_\_\_  
\_\_\_\_\_
18. In what order must the PTM control registers be configured? \_\_\_\_\_  
\_\_\_\_\_
19. Why is the order in the previous question required? \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

## Answers

12. The 6840 PTM must interface to the 8085 MPU as follows:
- The 6840 data lines  $D0-D7$  connect directly to the MPU data lines  $D0-D7$ .
  - The 6840  $RS0$ ,  $RS1$ , and  $RS2$  lines connect directly to the MPU address lines  $A0$ ,  $A1$ , and  $A2$  respectively.
  - The 6840  $\overline{CS0}$ , and  $CS1$  lines connect to the MPU address lines  $A3-A15$  using an address decoder.
  - The 6840  $R/\overline{W}$  and  $\overline{RESET}$  lines connect directly to the same MPU lines. ( $R/\overline{W}$  connects to  $\overline{MEMW}$ ).
  - The 6840 enable, or  $E$ , line connects directly to the MPU clock (CLK) line.
  - The 6840  $\overline{IRQ}$  line is connected to any of the MPU interrupt lines.
13. When the PTM  $R/\overline{W}$  line is high, a read operation is being performed by the MPU. The read only registers in the PTM are the three counter registers and the status register. Thus, one of these registers must be selected.
14. Once the PTM  $R/\overline{W}$  line determines the read/write register category, the exact PTM register that is selected is determined by the three register select, or  $RS$ , lines of the PTM, and bit 0 of the timer #2 control register.

15.

| Address    | Read             | Write                                 |
|------------|------------------|---------------------------------------|
| 80H        | None             | Timer #1 or Timer #3 Control Register |
| 81H        | Status Register  | Timer #2 Control Register             |
| 82H<br>83H | Timer #1 Counter | Timer #1 Latch                        |
| 84H<br>85H | Timer #2 Counter | Timer #2 Latch                        |
| 86H<br>87H | Timer #3 Counter | Timer #3 Latch                        |

16. The two major steps required to initialize the PTM are: write a beginning count value to the timer latches, and write a control word to the timer control registers.
17. Since the timer latches are 16-bit registers, you will normally use two MVI and two out instructions to transfer two bytes of data to the PTM.
18. The PTM control registers must be configured in the following order:
  - Write a control word to control register #3.
  - Write a control word to control register #2.
  - Write a control word to control register #1.
19. The order in the previous question is required since, after RESET, bit 0 of control register #2 is cleared. This accesses control register #3 at address XXX0. Therefore, control register #3 is first configured, then control register #2. While configuring control register #2, you must set bit 0 to access control register #1 at address XXX0. Control register #1 is then configured. Finally, when you configure control register #1, you must clear bit 0 to start the timers and complete the initialization routine.

## USING THE PTM

You should now understand how to interface, address, and initialize the PTM. As you are aware, programmable timers can be used to provide several different timing tasks. These include; continuous waveform and one-shot pulse generation, interval measurement, pulse measurement, and period measurement. Recall that these various timing tasks are grouped into timer output mode tasks and timer input mode tasks.

In this section, you will learn how to use the timer to provide each of the above mentioned tasks.

### Timer Output Mode Tasks

The timer output mode tasks include *interrupt generation*, *continuous waveform generation*, and *one-shot pulse generation*. To perform each of the output mode tasks, you must store a beginning count value to the timer latches. The control registers are then configured to provide the desired output mode task. When the final time-out, or TO, point is reached, output logic is generated by the PTM.

#### Interrupt Generation

To perform this task, you must connect the PTM  $\overline{IRQ}$  line directly to any of the MPU interrupt input lines. With the 8085 MPU, the PTM  $\overline{IRQ}$  line is connected directly to the 8085 RST5.5, 6.5, or 7.5 interrupt input lines as shown in Figure 5-22.

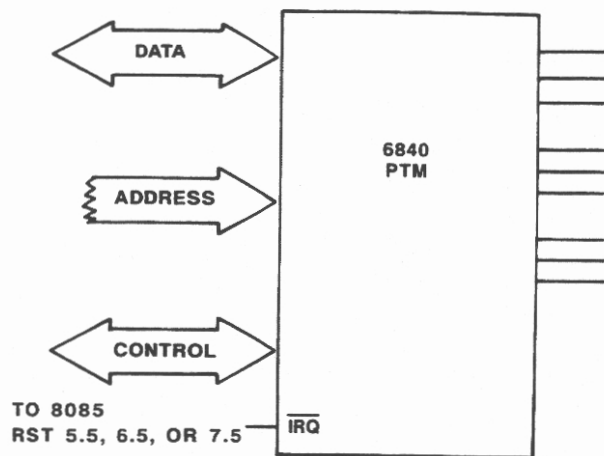


Figure 5-22  
Interrupt generation mode.

The PTM timer latches are loaded with a beginning count value, the control registers are configured to provide interrupt generation, and the timers are started. When the time-out point is reached, the PTM  $\overline{IRQ}$  line will go low and generate an interrupt to the MPU. This timing task is used to replace software time delays that, without the PTM, must be performed by the MPU. Using a PTM to provide the delay allows the MPU to be free of the timing task during the timed interval.

Using one PTM timer section, delays of up to 65,536 seconds, or approximately 18 hours, can be provided with a 1 Hz external +1 clock. With a 60 Hz external +1 clocking source, one timer section can provide delays up to 18 minutes. In the internal clocking mode, one timer section can provide delays up to 65.5 milliseconds, assuming a 1 MHz internal +1 clock. In any event, *the timed interval, in seconds, is always equal to the beginning timer count value plus 1, divided by the timer clocking frequency.*

Let's assume, for example, that you wish to provide a 10-second time delay using timer #3 of the PTM. The easiest way to provide such a delay is to externally clock timer #3 from a 60 Hz clocking source. Recall that the time delay is always the beginning count value plus 1, divided by the timer clock frequency, or:

$$\text{time delay} = \frac{(\text{beginning count value} + 1)}{\text{clock frequency}}$$

Using this equation, the required count value plus 1 is equal to the desired time delay multiplied by the clock frequency, or:

$$(\text{beginning count value} + 1) = (\text{time delay}) \times (\text{clock frequency})$$

In our example,

$$(\text{beginning count value} + 1) = (10 \text{ sec.}) \times (60 \text{ Hz})$$

$$(\text{beginning count value} + 1) = 600_{10}$$

$$\text{beginning count value} = 600_{10} - 1$$

$$= 599_{10}$$

This is the value that must be stored in the timer #3 latch during the PTM initialization routine to provide a 10-second delay using a 60 Hz external +1 clock. However, the value obtained above is a base ten value and must be converted to hexadecimal. The equivalent hexadecimal value is 257H.

Refer to the circuit shown in Figure 5-17 and assume that a 60 Hz clock signal, is applied to the timer #3 external clock line. Now, the program listed in Figure 5-23 will initialize the PTM to provide a 10-second time delay using timer #3. The program first stores the beginning count value, or 0257H, to the timer #3 latch. Control register #3 is then configured by storing the hex value 60H to port address 50H. This value configures control register #3 as illustrated in Figure 5-24. You might note that the one-shot mode is used to provide the time delay interrupt task. In addition, the timer #3 interrupt must be enabled.

| SOURCE CODE                                  | COMMENTS                                                  |
|----------------------------------------------|-----------------------------------------------------------|
| MVI A,02H<br>OUT 56H<br>MVI A,57H<br>OUT 57H | STORE BEGINNING COUNT VALUE,<br>0257H, TO TIMER #3 LATCH  |
| MVI A,60H<br>OUT 50H                         | CONFIGURE CONTROL REGISTER #3<br>FOR INTERRUPT GENERATION |
| MVI A,01H<br>OUT 51H                         | START THE TIMER OPERATION                                 |
| MVI A,00H<br>OUT 50H                         |                                                           |

Figure 5-23

Program for time delay and interrupt generation using timer #3.

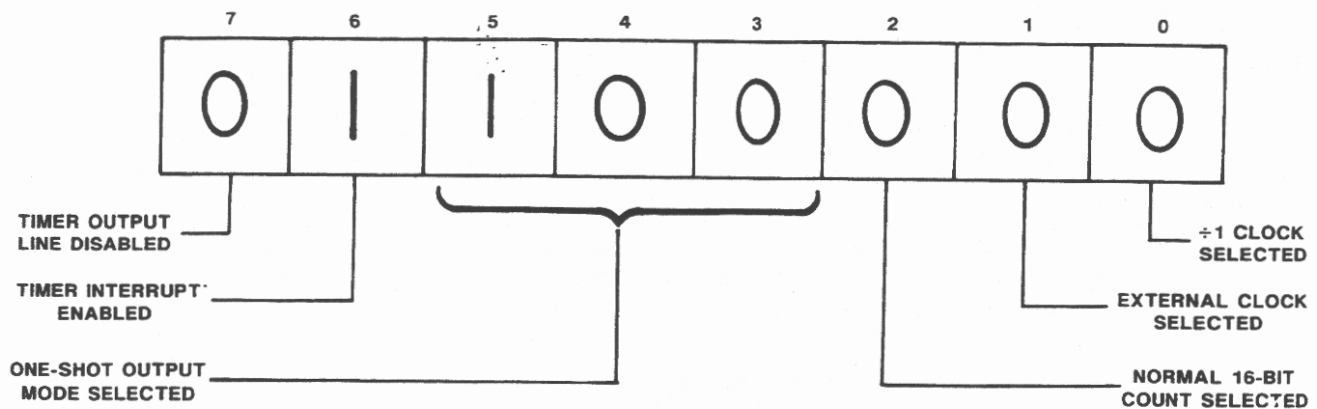


Figure 5-24

Control register #3 configured for time delay, interrupt generation.

The remaining part of the program in Figure 5-23 sets bit 0 of control register #2 at address 51H so that control register #1 can be subsequently accessed at address 50H. Bit 0 of control register #1 is then cleared to start the time delay operation.

After 10 seconds, timer #3 will time-out and generate an interrupt to the MPU. In the meantime, the MPU is free to perform other, more important tasks.

Using this same program, a 10-minute delay could be provided by simply driving timer #3 with a 1 Hz external clock rather than a 60 Hz external clock. By changing the external clock frequency and/or the beginning count value, many different time delays can be provided with just one PTM timer section.

For very long time delays, the PTM timer sections must be cascaded together as shown in Figure 5-25. In this arrangement, the timer #3 output line clocks timer #2, and the timer #2 output line clocks timer #1. Timer #3 can be clocked internally, or clocked from an external clocking source. The only interrupt enabled must be the timer #1 interrupt since the time delay is not completed until timer #1 times-out. Timer #3 and timer #2 must be configured for the continuous output mode and timer #1 configured for either the continuous or the one-shot mode of operation. When timer #1 is configured in the continuous mode, the interrupt would be periodically generated each time timer #1 times-out. When configured in the one-shot mode, timer #1 will only time-out once, and thus provide only one time delay interrupt.

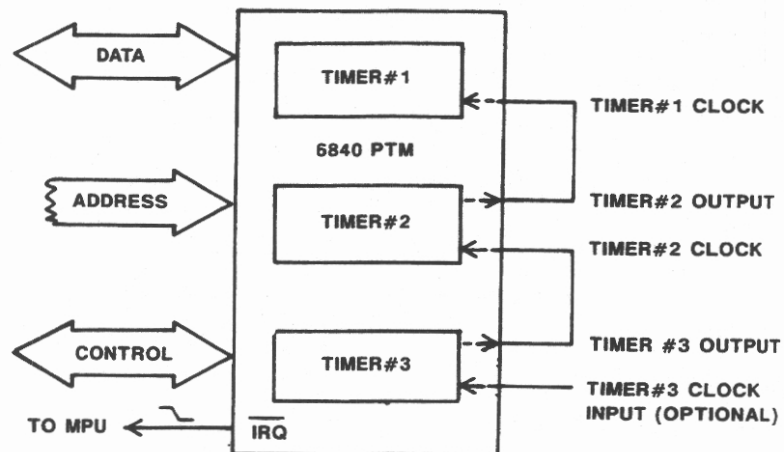


Figure 5-25

Cascading the PTM timers for long time delays.

## Continuous Waveform Generation

The PTM timers can be configured to generate a TTL waveform of variable frequency and duty cycle. A single timer section is used to generate medium to high output frequencies, or the timers can be cascaded together as shown in Figure 5-25 to obtain very low output frequencies. The output frequency is dependent upon the beginning count value in the timer latch(es). The larger the beginning count value, the lower the output frequency. Conversely, the smaller the beginning count value, the higher the output frequency. Of course, you can also change the output frequency by changing the timer clock rate. A high clock rate will provide a high output frequency, given a constant beginning count value.

There are two types of continuous output modes of operation available to the PTM. One mode, called the **continuous 16-bit mode**, generates a 50% duty cycle output waveform as shown in Figure 5-26. Each time the timer times-out, a logic transition is provided on the timer's output line. Thus, it takes two time-out periods to generate one cycle of the output waveform. With the 16-bit mode, both time-out (TO) periods are equal, and a 50% duty cycle results. As shown in Figure 5-26, the time-out period is equal to the beginning count value plus 1, divided by the clock frequency. The period of the output waveform is then two times the time-out period. The frequency of the output waveform is the reciprocal of its period.

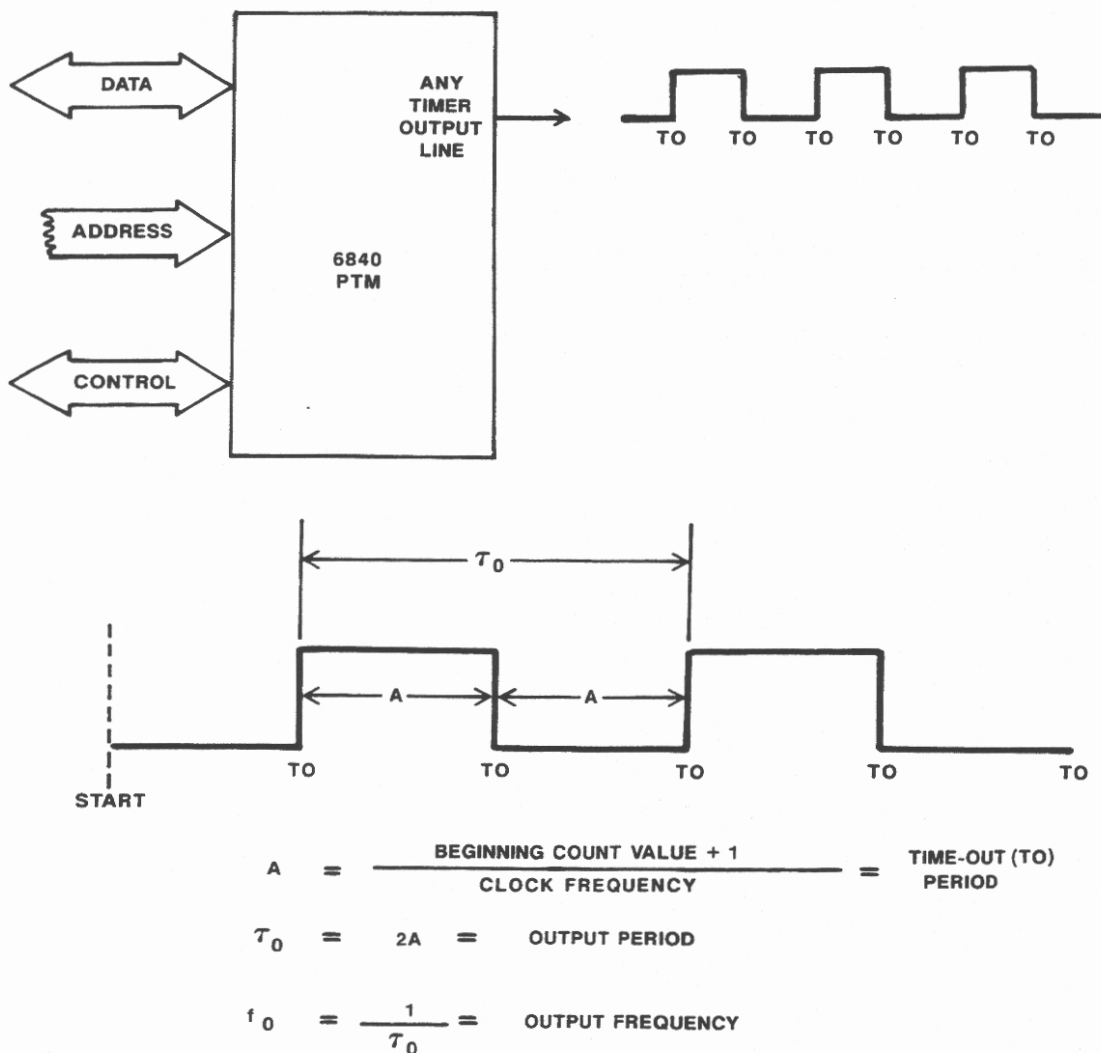
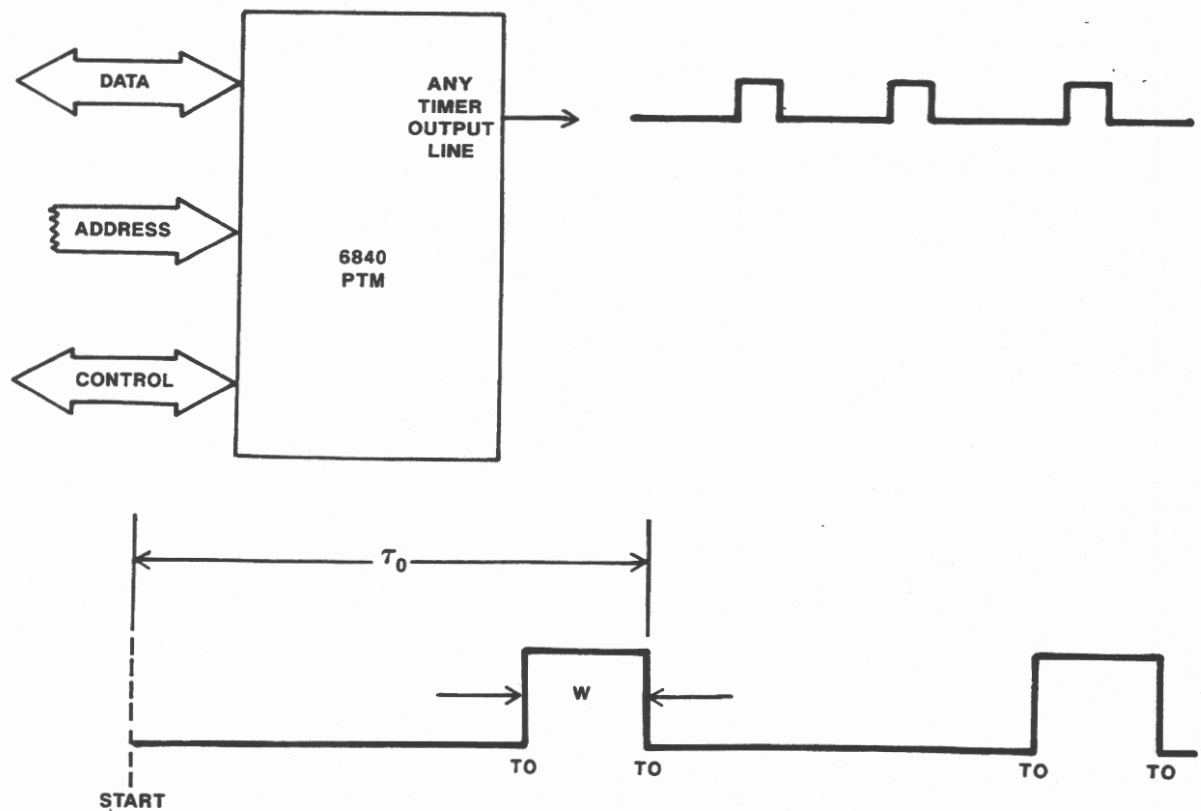


Figure 5-26  
Continuous 16-bit output mode operation.



The other continuous mode of operation is called the **continuous dual 8-bit mode**. This mode is used to generate a continuous output waveform with a variable duty cycle as shown in Figure 5-27. The term dual 8-bit is used since the 16-bit counter is divided into two 8-bit counters: a least significant byte, or LSB, counter; and a most significant byte, or MSB, counter. A 16-bit beginning count value is therefore divided into an 8-bit MSB count and an 8-bit LSB count.



$$W = \frac{\text{LSB COUNT} + 1}{\text{CLOCK FREQUENCY}}$$

$$T_0 = \frac{(\text{MSB COUNT} + 1) \times (\text{LSB COUNT} + 1)}{\text{CLOCK FREQUENCY}} = \text{OUTPUT PERIOD}$$

$$f_0 = \frac{1}{T_0} = \text{OUTPUT FREQUENCY}$$

$$\text{DUTY CYCLE} = \frac{W}{T_0} \times 100\% = \frac{1}{(\text{MSB COUNT} + 1)} \times 100\%$$

Figure 5-27  
Continuous dual 8-bit output operation.

As shown in Figure 5-27, the period of the output waveform is always the LSB count plus 1, times the MSB count plus 1, divided by the clock frequency. The output frequency of the waveform is then the reciprocal of this value.

The duty cycle of the dual 8-bit output waveform is determined by the beginning MSB count. From Figure 5-27, the output waveform duty cycle is  $\frac{w}{\tau_o} \times 100\%$ .

Algebraically, this reduces to approximately  $\frac{1}{(\text{MSB count} + 1)} \times 100\%$ . Thus, the duty cycle is equal to the reciprocal of the MSB count plus 1, times 100%. This means that the output waveform duty cycle is independent of the beginning LSB count and the clock frequency.\*

Now, let's look at a program that will initialize the PTM for both the 16-bit and dual 8-bit continuous operating modes. We will use timer #3 to generate a continuous waveform with a 50% duty cycle, and timer #2 to generate a continuous waveform that does not have a 50% duty cycle. The PTM will thus provide two separate continuous waveforms as shown in Figure 5-28. Both timers will be clocked from the internal PTM clock. Using the 8085 microprocessor module, the ETW3800 Trainer operates on a clock frequency of approximately 1.84 MHz. We will, therefore, assume this clock frequency in our examples.

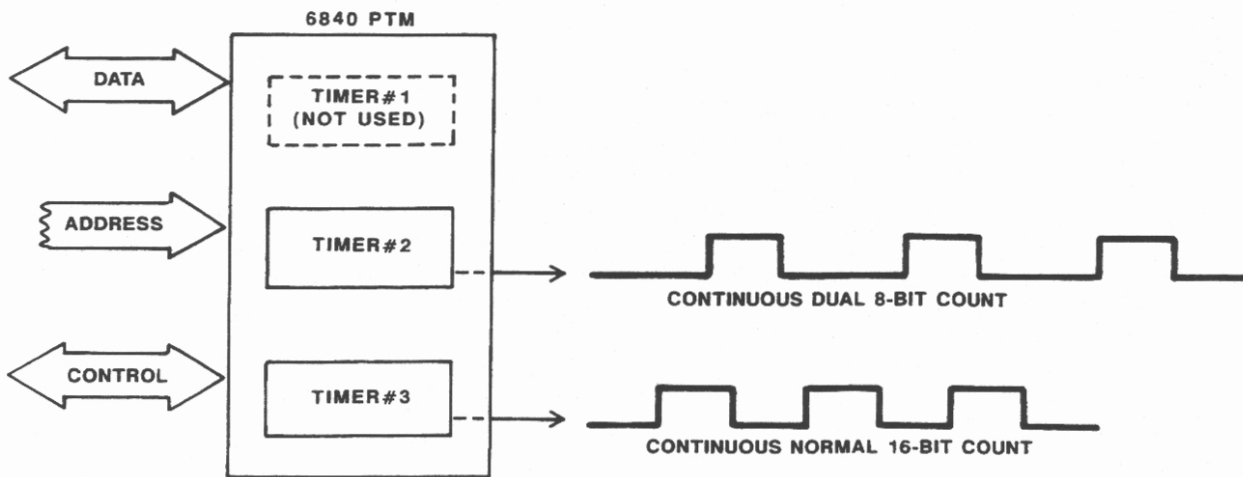


Figure 5-28

Continuous output waveform generation: timer #3 continuous 16-bit output and timer #2 dual 8-bit output. Timer #1 is not used in this example.

\*Assuming a beginning LSB count of at least  $OA_{16}$ .

| SOURCE CODE                                  | COMMENTS                                                                |
|----------------------------------------------|-------------------------------------------------------------------------|
| MVI A,03H<br>OUT 56H<br>MVI A,97H<br>OUT 57H | STORE BEGINNING COUNT VALUE, TO<br>TIMER #3 LATCH                       |
| MVI A,04H<br>OUT 54H<br>MVI A,FFH<br>OUT 55H | STORE BEGINNING COUNT VALUE TO THE<br>TIMER #2 LATCH                    |
| MVI A,82H<br>OUT 50H                         | CONFIGURE CONTROL REGISTER #3 FOR<br>THE CONTINUOUS 16-BIT OUTPUT MODE. |
| MVI A,87H<br>OUT 51H                         | CONFIGURE CONTROL REGISTER #2 FOR<br>THE DUAL 8-BIT OUTPUT MODE.        |
| MVI A,00H<br>OUT 50H                         | START TIMERS                                                            |

Figure 5-29

Program to configure timer #3 for continuous 16-bit output and timer #2 for continuous dual 8-bit output.

The program listed in Figure 5-29 will initialize the PTM for the given task. Again, we have assumed that the PTM is assigned to port addresses 50H through 5FH as before. The program begins by storing the hex value 0397H to the timer #3 latch. Timer #3 is to be configured to provide a continuous output waveform with a 50% duty cycle. Therefore, the beginning count value of 0397H will generate a time-out period of:

$$\begin{aligned}
 A &= \frac{(\text{beginning count value} + 1)}{(\text{clock frequency})} \\
 &= \frac{(0397 + 1)_{16}}{1.84 \text{ MHz}} \\
 &= \frac{0398_{16}}{1,840,000_{10}} \\
 &= \frac{920_{10}}{1,840,000_{10}} \\
 &= .0005 \text{ seconds}
 \end{aligned}$$

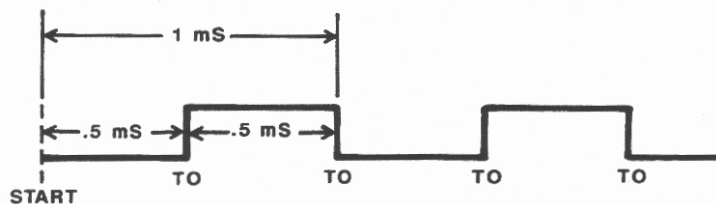
The output waveform period is equal to twice the time-out period. Therefore, the output waveform period is:

$$\begin{aligned}\tau_o &= 2A \\ &= 2 \times (.0005 \text{ seconds}) \\ &= .001 \text{ seconds}\end{aligned}$$

The output frequency is then the reciprocal of this output period, or:

$$\begin{aligned}f_o &= \frac{1}{\tau_o} \\ &= \frac{1}{.001 \text{ seconds}} \\ &= 1000 \text{ Hz}\end{aligned}$$

Thus, the continuous output waveform in Figure 5-30 is generated by timer #3.



|                           |   |                    |
|---------------------------|---|--------------------|
| TIMER CLOCK FREQUENCY     | = | 1.84 MHz           |
| BEGINNING COUNT VALUE     | = | 0397 <sub>16</sub> |
| TIME-OUT PERIOD           | = | .5 mS              |
| OUTPUT WAVEFORM PERIOD    | = | 1 mS               |
| OUTPUT WAVEFORM FREQUENCY | = | 1 KHz              |

Figure 5-30

The continuous output waveform generated by timer #3 with the PTM initialization program listed in Figure 5-29.

Next, the program stores the hex value 04FFH to the timer #2 latch. Timer #2 is to be configured to provide a dual 8-bit continuous output waveform. The beginning count value of 04FFH will generate a period of:

$$\begin{aligned}
 \tau_o &= \frac{(\text{MSB count} + 1) \times (\text{LSB count} + 1)}{\text{clock frequency}} \\
 &= \frac{(04 + 1)_{16} \times (\text{FF} + 1)_{16}}{1.84 \text{ MHz}} \\
 &= \frac{05_{16} \times 100_{16}}{1,840,000_{10}} \\
 &= \frac{05_{10} \times 256_{10}}{1,840,000_{10}} \\
 &= \frac{1280_{10}}{1,840,000_{10}} \\
 &= .0007 \text{ seconds (approx.)}
 \end{aligned}$$

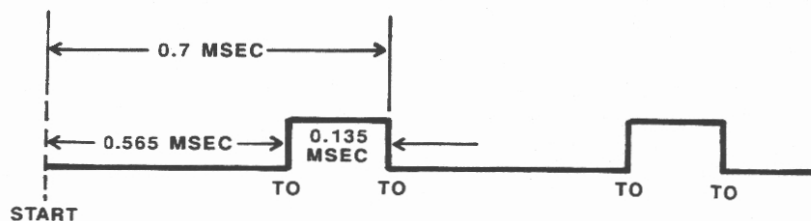
The output frequency is then the reciprocal of this output period, or:

$$\begin{aligned}
 f_o &= \frac{1}{\tau_o} \\
 &= \frac{1}{.0007 \text{ sec.}} \\
 &= 1429 \text{ Hz}
 \end{aligned}$$

However, the output waveform does not necessarily have a 50% duty cycle, since timer #2 is to be configured for the dual 8-bit continuous mode. To calculate the duty cycle in the dual 8-bit mode, you must find the reciprocal of the beginning MSB count plus 1 and multiply this value by 100%. The duty cycle in this example will therefore be:

$$\begin{aligned}
 \text{Duty cycle} &= \frac{1}{(\text{MSB count} + 1)} \times 100\% \\
 &= \frac{1}{04_{16} + 1} \times 100\% \\
 &= \frac{1}{05_{16}} \times 100\% \\
 &= \frac{1}{5_{10}} \times 100\% \\
 &= .2_{10} \times 100\% \\
 &= 20\%
 \end{aligned}$$

A duty cycle of 20% means that the output waveform will be high for 20% of its period. Thus, the continuous output waveform shown in Figure 5-31 is generated by timer #2.



TIMER CLOCK FREQUENCY = 1.84 MHz  
 BEGINNING COUNT VALUE =  $04FF_{16}$   
 $W = 0.135$  MILLISECONDS  
 OUTPUT WAVE FORM PERIOD = 0.7 MILLISECONDS  
 OUTPUT WAVEFORM FREQUENCY = 1429 Hz  
 DUTY CYCLE = 20%

**Figure 5-31**

The continuous output waveform generated by timer #2 with the PTM initialization program listed in Figure 5-29.

The next two parts of the PTM initialization program in Figure 5-29 configure the timer #3 and timer #2 control registers to perform the required task. The control register configurations provided by this program are shown in Figure 5-32. Note that, aside from the bit 0 status of each control register, the only difference between the timer #3 and timer #2 control registers is the bit 2 status. Recall that bit 2 is used to select between the normal 16-bit count mode and the dual 8-bit count mode. In the timer #3 control register, bit 2 is cleared to select the normal 16-bit count mode. In the timer #2 control register, bit 2 is set to select the dual 8-bit count mode. You should now be familiar with the remaining bit status requirements and understand the remaining control register bit configurations shown in Figure 5-32.

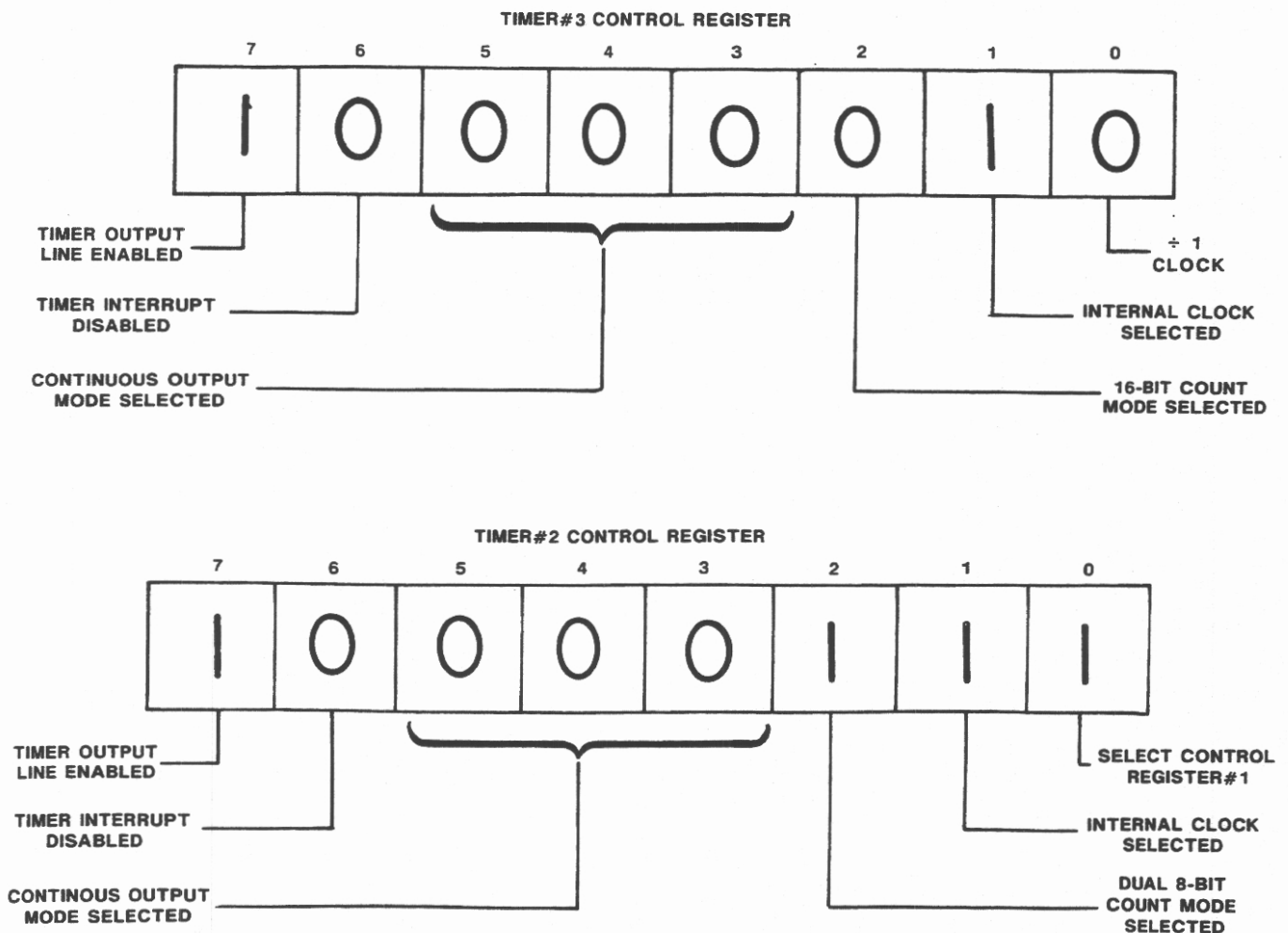


Figure 5-32  
Timer #3 and timer #2 control register configurations  
provided by the PTM initialization program listed  
in Figure 5-29.

## ONE-SHOT PULSE GENERATION

The one-shot pulse generation mode is similar to the continuous mode in that you can configure the timers for a normal 16-bit count or a dual 8-bit count. Selection of either count is again provided by bit 2 of the respective control register.

The one-shot 16-bit output pulse is illustrated in Figure 5-33. When the timer is started, a one-count delay, equal to the clock period, takes place before the one-shot pulse is generated. After the one-count delay, the timer output line will go high for an amount of time equal to the beginning count value divided by the clock frequency. For example, suppose the PTM is initialized to configure timer #2 for the one-shot, 16-bit pulse generation mode using the program listed in Figure 5-34. We will assume a 1 Hz external clocking pulse for this example. With the program listed in Figure 5-34 the beginning count value stored to the timer #2 latch is 000AH. The program then configures the timer #2 control register as shown in Figure 5-35.

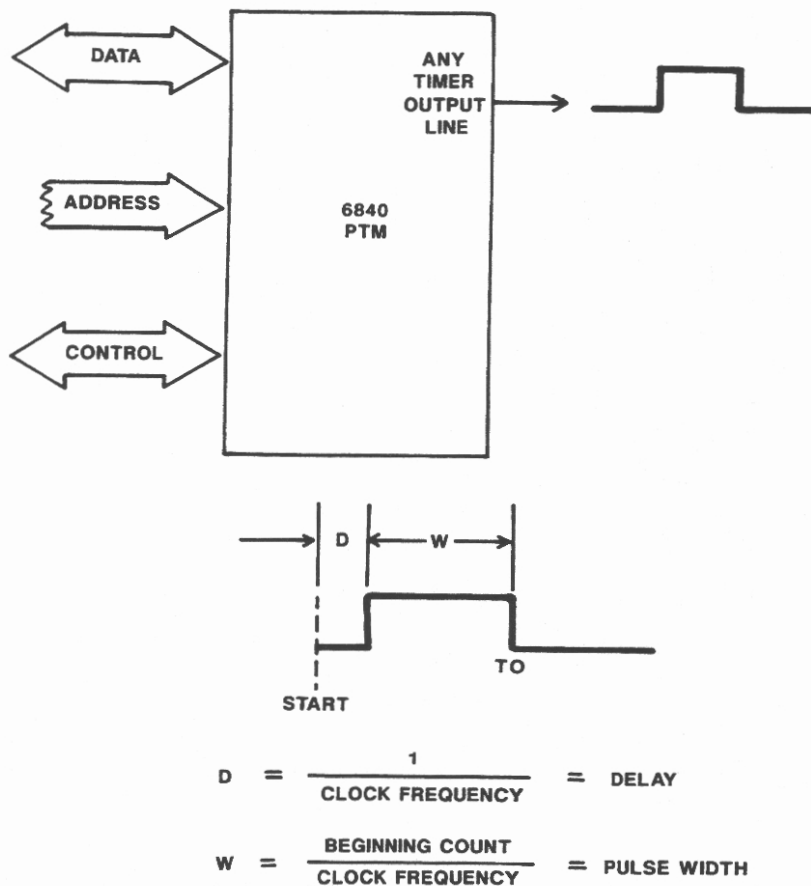


Figure 5-33  
One-shot 16-bit output mode operation.



| SOURCE CODE                                  | COMMENTS                                                                    |
|----------------------------------------------|-----------------------------------------------------------------------------|
| MVI A,00H<br>OUT 54H<br>MVI A,0AH<br>OUT 55H | STORE BEGINNING COUNT VALUE, 000AH TO THE TIMER #2 LATCH                    |
| MVI A,A1H<br>OUT 51H                         | CONFIGURE CONTROL REGISTER #2 FOR THE ONE-SHOT 16-BIT PULSE GENERATION MODE |
| MVI A,00H<br>OUT 50H                         | START THE TIMERS                                                            |

Figure 5-34  
Program to configure timer #2 for one-shot 16-bit pulse generation.

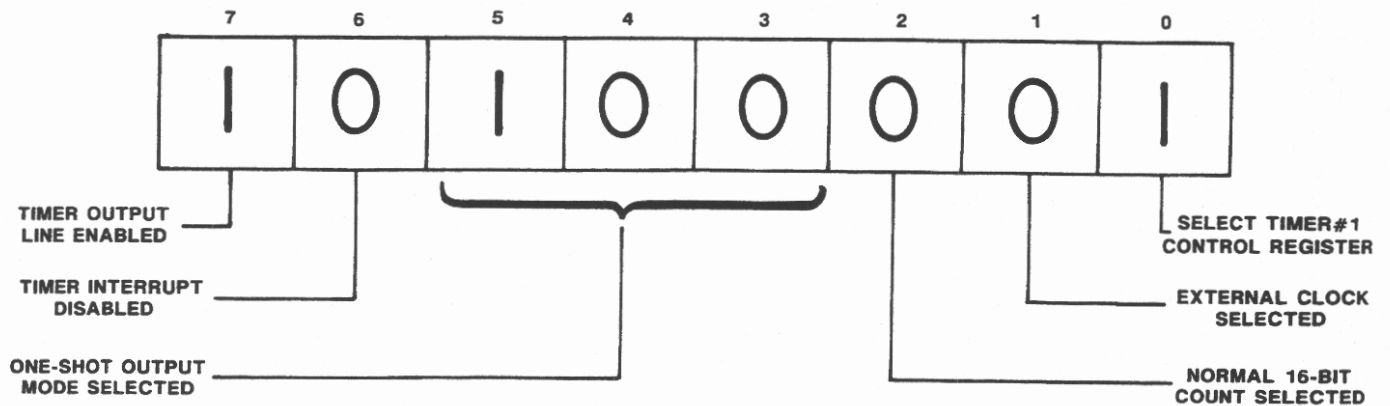


Figure 5-35  
Timer #2 control register configuration provided by the PTM initialization program listed in Figure 5-34.

Once the timer is started, there will be a delay before the pulse is generated. This delay, or  $D$ , is equal to one clock period. With a 1 Hz external clocking source, the delay is 1 second. After the 1-second delay, the timer #2 output line will go high for an amount of time equal to the beginning count value divided by the external clock frequency. In this example, the one-shot high pulse would last for:

$$\begin{aligned}
 W &= \frac{\text{beginning count}}{\text{clock frequency}} \\
 &= \frac{000A_{16}}{1 \text{ Hz}} \\
 &= \frac{10_{10}}{1 \text{ Hz}} \\
 &= 10 \text{ seconds}
 \end{aligned}$$

Thus, after a 1-second delay, the timer #2 output line will go high and remain high for 10 seconds, before returning low as shown in Figure 5-36. The timer must then be reinitialized by the MPU to generate another one-shot pulse.

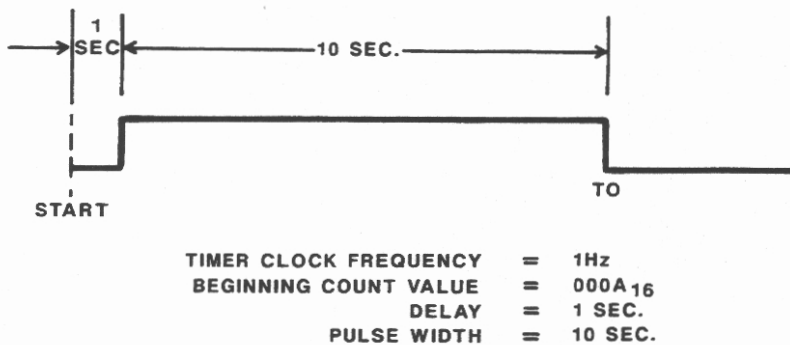


Figure 5-36

The one-shot pulse generated by timer #2 with the PTM initialization program listed in Figure 5-34.

The one-shot, dual 8-bit mode output pulse is illustrated in Figure 5-37. This differs from the 16-bit mode in that the amount of time before the pulse is generated depends on the beginning count value. Furthermore, the pulse width is the LSB count divided by the clock frequency.

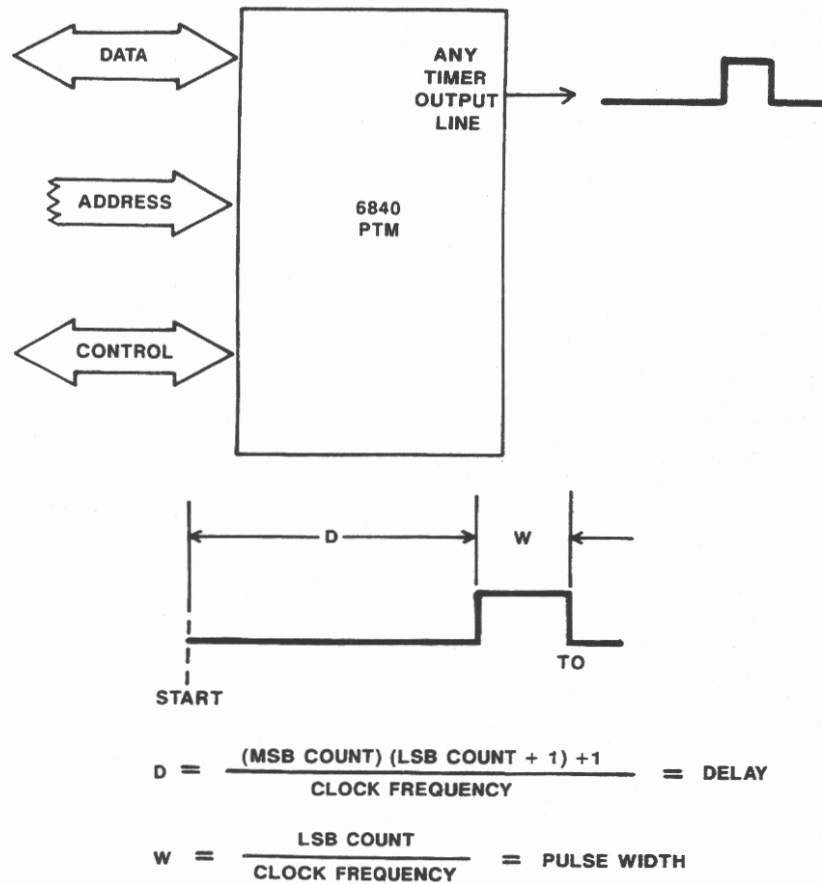


Figure 5-37  
One-shot dual 8-bit output pulse mode.

Suppose the PTM is initialized to configure timer #2 for the one-shot dual 8-bit pulse generation mode using the program listed in Figure 5-38 and the control register configuration shown in Figure 5-39. We will assume a 60 Hz external clocking source for this example. Once the timer is started, there will be a delay of:

$$\begin{aligned}
 D &= \frac{(\text{MSB count})(\text{LSB count} + 1) + 1}{\text{clock frequency}} \\
 &= \frac{(06_{16}) \times (B4_{16} + 1) + 1}{60_{10} \text{ Hz}} \\
 &= \frac{(06_{16}) \times (B5_{16}) + 1}{60_{10} \text{ Hz}} \\
 &= \frac{43E_{16} + 1}{60_{10} \text{ Hz}} \\
 &= \frac{43F_{16}}{60_{10} \text{ Hz}} \\
 &= \frac{1087_{10}}{60_{10} \text{ Hz}} \\
 &= 18 \text{ seconds}
 \end{aligned}$$

| SOURCE CODE                                  | COMMENTS                                                                        |
|----------------------------------------------|---------------------------------------------------------------------------------|
| MVI A,06H<br>OUT 54H<br>MVI A,B4H<br>OUT 55H | STORE BEGINNING COUNT VALUE, 06B4H, TO THE TIMER #2 LATCH                       |
| MVI A,A5H<br>OUT 51H                         | CONFIGURE CONTROL REGISTER #2 FOR THE ONE-SHOT DUAL 8-BIT PULSE GENERATION MODE |
| MVI A,00H<br>OUT 50H                         | START THE TIMERS                                                                |

Figure 5-38

Program to configure timer #2 for one-shot dual 8-bit pulse generation.

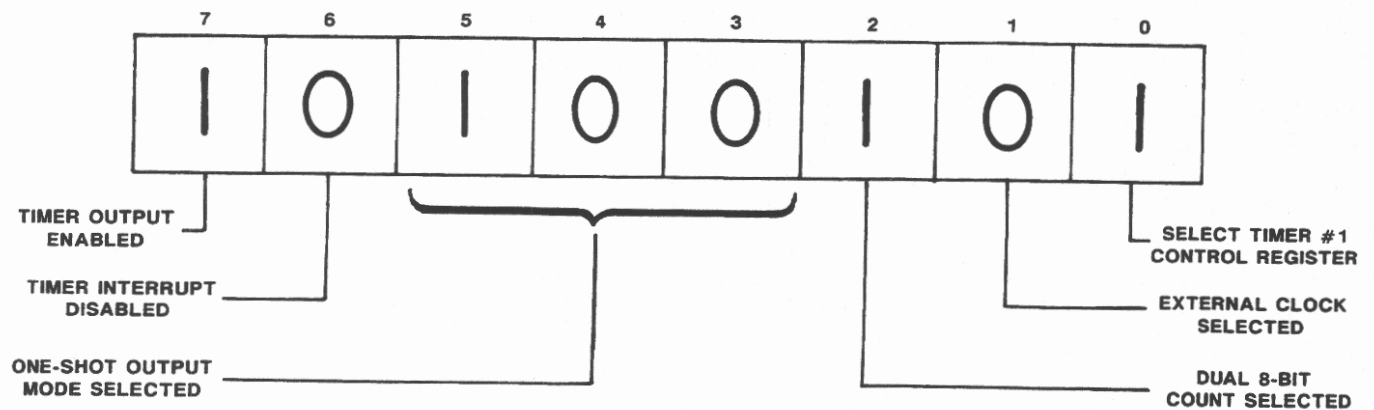


Figure 5-39

Timer #2 control register configuration provided by the PTM initialization program listed in Figure 5-38.

After this 18-second delay, the timer #2 output line will go high for an amount of time equal to the LSB count divided by the clock frequency. In this example, the one-shot high pulse would last for:

$$\begin{aligned}
 W &= \frac{\text{LSB count}}{\text{clock frequency}} \\
 &= \frac{B4_{16}}{60_{10} \text{ Hz}} \\
 &= \frac{180_{10}}{60_{10} \text{ Hz}} \\
 &= 3 \text{ seconds}
 \end{aligned}$$

Thus, after an 18-second delay, the timer #2 output line will go high and remain high for 3 seconds, before returning low as shown in Figure 5-40. The timer must then be initialized again by the MPU if another one-shot pulse is to be generated.

Longer delays and pulse widths can be achieved by cascading the PTM timers together. Delays and pulse widths up to many *years* can be created using cascaded timers.

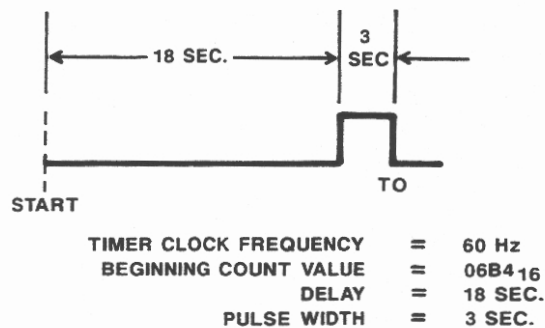


Figure 5-40

The one-shot dual 8-bit pulse generated by timer #2 with the PTM initialization program listed in Figure 5-38.

## Timer Input Mode Tasks

The timer input mode tasks include *interval measurement*, *pulse measurement*, and *period measurement*. We will discuss period measurement first, since the other two are simply special cases of the period measurement task.

## PERIOD MEASUREMENT

The frequency of an unknown input waveform is determined using the period measurement mode. In all of the input mode tasks, the input signal is applied to the respective timer input gate line.

The timer is initialized by configuring its control register for the input period measurement mode. Once initialized, a high-to-low transition on the timer's gate line will start the timer counter. The next consecutive high-to-low transition will stop the counter. The applied signal period is then determined by subtracting the final count value from the beginning count value, and dividing the difference by the clock frequency. Once the period of the input signal is determined, the frequency is found by taking the reciprocal of the period. The period measurement process is illustrated in Figure 5-41.

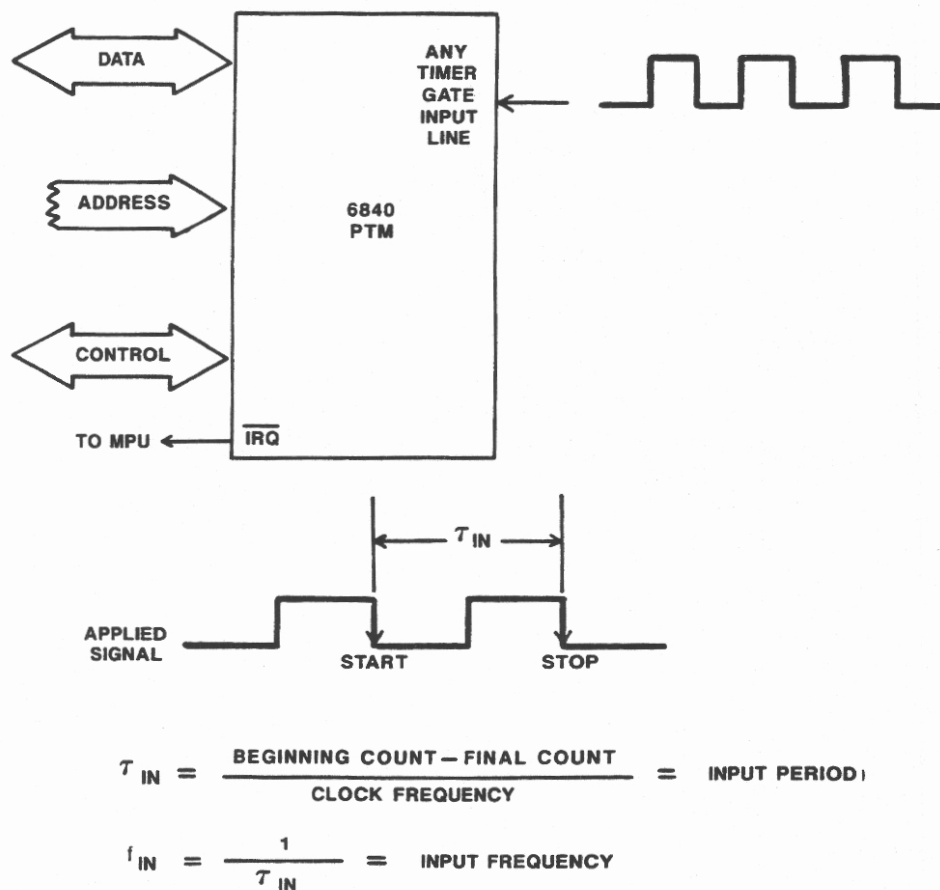


Figure 5-41

Period measurement input mode operation.

For example, suppose timer #3 is to be used in the period measurement mode to measure the frequency of an unknown input signal. We will assume that an internal clock frequency of 1.84 MHz is used to clock timer #3 and that the timer is mapped to addresses 50H through 57H as before. The required PTM initialization program is listed in Figure 5-42.

The program in Figure 5-42 first stores the hex value  $4A_{16}$  to the timer #3 control register. This will configure timer #3 as shown in Figure 5-43. Note that the timer interrupt is enabled and the period measurement mode is selected. After configuring the timer #3 control register, the program starts the timer. A beginning count value is not stored to the timer #3 latch. Thus, the beginning count value is automatically set to FFFFH.

| SOURCE CODE                                  | COMMENTS                                                            |
|----------------------------------------------|---------------------------------------------------------------------|
| MVI A,4AH<br>OUT 50H                         | CONFIGURE CONTROL REGISTER #3 FOR THE PERIOD MEASUREMENT INPUT MODE |
| MVI A,01H<br>OUT 51H<br>MVI A,00H<br>OUT 50H | START                                                               |

Figure 5-42

Program to initialize timer #3 for the period measurement input mode.

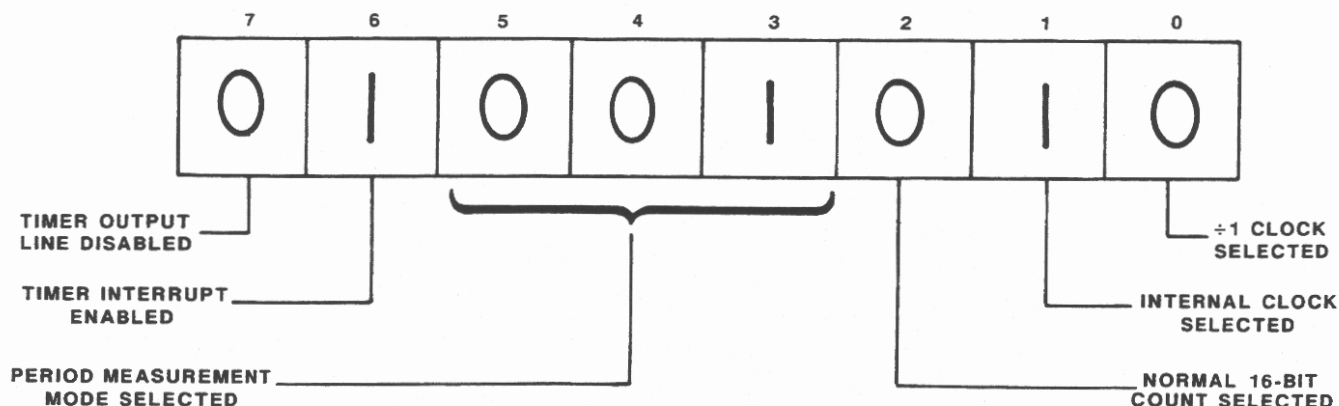


Figure 5-43

Control register #3 configuration provided by the initialization program listed in Figure 5-42.



Now suppose an unknown signal applied to the timer #3 gate input line starts the counter with a high-to-low transition. On the next consecutive high-to-low transition, the counter is stopped and the PTM generates an interrupt to the MPU. At this time, the MPU reads the counter's contents at port address 56H to obtain the final count.

Suppose the final count is found to be  $C400_{16}$ . The MPU must then determine the unknown input signal frequency. The input signal period is:

$$\begin{aligned}
 \tau_{in} &= \frac{(\text{beginning count}) - (\text{final count})}{\text{clock frequency}} \\
 &= \frac{FFFF_{16} - 88AF_{16}}{1.84 \text{ MHz}} \\
 &= \frac{7750_{16}}{1,840,000_{10}} \\
 &= \frac{30,544_{10}}{1,840,000_{10}} \\
 &= .0166 \text{ seconds}
 \end{aligned}$$

Then, the input signal frequency is:

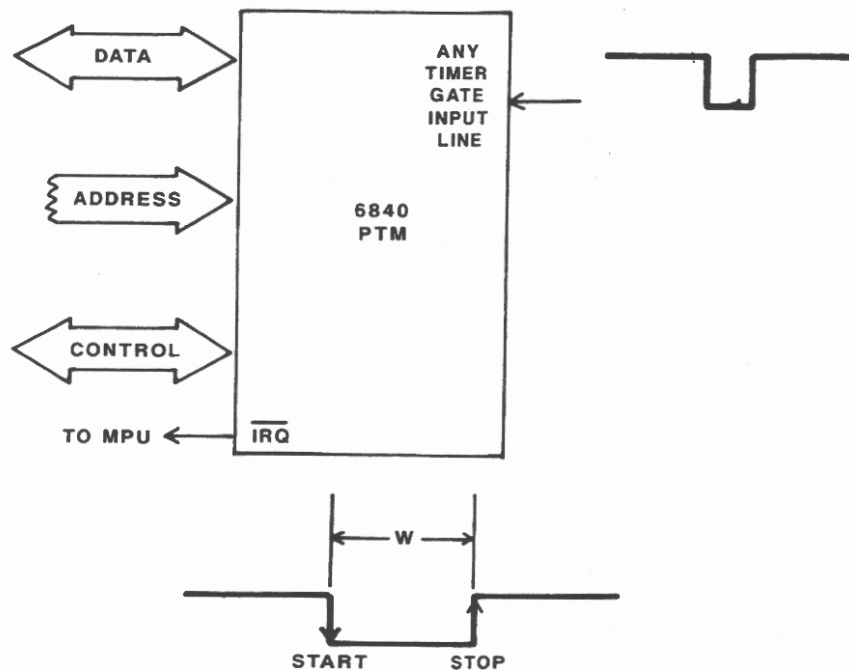
$$\begin{aligned}
 f_{in} &= \frac{1}{\tau_{in}} \\
 &= \frac{1}{.0166 \text{ seconds}} \\
 &= 60 \text{ Hz}
 \end{aligned}$$

The above calculations must be part of the PTM interrupt service routine.

The period measurement mode is also used to measure the time between two consecutive external events. After initialization, the first event simply starts the counter with a high-to-low transition on the timer's gate input line. The second event stops the counter. The time between events is then determined by subtracting the final count from the beginning count, and dividing the difference by the clock frequency.

## PULSE MEASUREMENT

Pulse measurement is a special case of period measurement. In the pulse measurement mode, the counter is started with a high-to-low transition on the timer's gate input line. Then, the counter is stopped with a subsequent low-to-high transition on the timer's gate input line. The pulse width is then determined by subtracting the final count value from the beginning count value, and dividing the difference by the clock frequency. The pulse width measurement process is illustrated in Figure 5-44.



$$W = \frac{\text{BEGINNING COUNT} - \text{FINAL COUNT}}{\text{CLOCK FREQUENCY}} = \text{PULSE WIDTH}$$

Figure 5-44

Pulse measurement input mode operation.

The program listed in Figure 5-45 can be used to initialize timer #3 for the pulse measurement mode. The only difference between this program and the period measurement initialization program previously discussed is the value that is stored in the timer #3 control register. The pulse measurement initialization program stores the hex value 5AH in the timer #3 control register to configure it for pulse measurement as shown in Figure 5-46. With this configuration, a high-to-low transition on the timer #3 gate input line starts the counter. When the input pulse makes a low-to-high transition, the counter is stopped and the PTM generates an interrupt to the MPU. The MPU must then read the final count from the timer #3 counter located at address 56H.

| SOURCE CODE                                  | COMMENTS                                                           |
|----------------------------------------------|--------------------------------------------------------------------|
| MVI A,5AH<br>OUT 50H                         | CONFIGURE CONTROL REGISTER #3 FOR THE PULSE MEASUREMENT INPUT MODE |
| MVI A,01H<br>OUT 51H<br>MVI A,00H<br>OUT 50H | START                                                              |

Figure 5-45

Program to initialize timer #3 for the pulse measurement input mode.

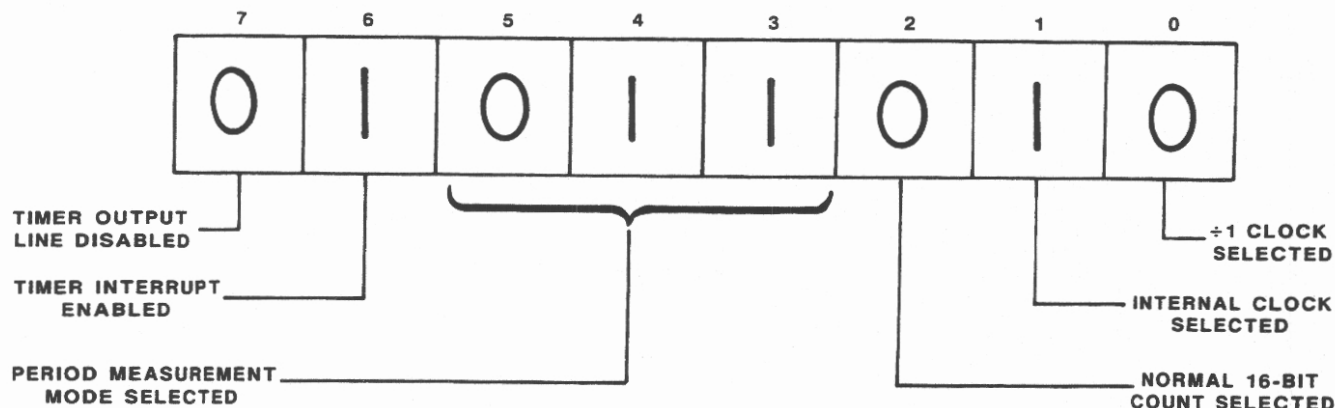


Figure 5-46

Control register #3 configuration provided by the PTM initialization program listed in Figure 5-45.

Suppose the final count is found to be FA3FH. The MPU must then determine the input pulse width as follows:

$$\begin{aligned} W &= \frac{(\text{beginning count}) - (\text{final count})}{\text{clock frequency}} \\ &= \frac{\text{FFFF}_{10} - \text{FA3F}_{16}}{1.84 \text{ MHz}} \\ &= \frac{05C0_{16}}{1.84 \text{ MHz}} \\ &= \frac{1472_{10}}{1,840,000_{10}} \\ &= .0080 \text{ seconds} \end{aligned}$$

## Self-Test Review

20. What two things control the length of a timed interval, and how must you calculate the time-out period of a timer configured in the interrupt generation mode? \_\_\_\_\_  
\_\_\_\_\_
21. Suppose a 60 Hz external clocking frequency is used to clock timer #1. What beginning count value must be stored in the timer #1 latch to provide a 5-minute time delay? \_\_\_\_\_  
\_\_\_\_\_
22. How can very long time delays be created using the PTM? \_\_\_\_\_  
\_\_\_\_\_
23. What are the two continuous output modes of operation available with the 6840 PTM, and how do these two modes differ? \_\_\_\_\_  
\_\_\_\_\_
24. Given an internal clocking frequency 920 kHz, and timer #2 configured in the continuous 16-bit count mode. If the hex value 0355H is stored to the timer #2 latch during initialization, what output frequency will be generated on the timer #2 output line? \_\_\_\_\_  
\_\_\_\_\_
25. Suppose timer #2 is configured as in question 24, but the dual 8-bit count mode is used. How does this affect the output frequency? \_\_\_\_\_  
\_\_\_\_\_
26. What is the duty cycle of the timer #2 output wave form in question 25? \_\_\_\_\_  
\_\_\_\_\_
27. Timer #3 is to be used to generate a one-shot pulse after a 10-minute delay. The width of the pulse is to be 2 seconds. What hex value must be stored to the timer #3 latch to provide the desired pulse using a 1 Hz external clock? \_\_\_\_\_  
\_\_\_\_\_
28. What hex value must the timer #3 control register contain to provide the output in question 27? \_\_\_\_\_  
\_\_\_\_\_



## Answers

20. The two things that control the length of a timed interval are the beginning timer count value and the timer clocking frequency.

The time-out period of a timer configured in the interrupt generation mode is calculated by dividing the timer clocking frequency into the beginning count value plus 1.

21. The time delay period is equal to the beginning count value plus 1, divided by the clock frequency, or

$$\text{time delay} = \frac{(\text{beginning count} + 1)}{(\text{clock frequency})}$$

Therefore,

$$\begin{aligned}\text{beginning count} + 1 &= (\text{time delay}) \times (\text{clock frequency}) \\ &= (5 \text{ min.}) \times (60 \text{ HZ}) \\ &= (300 \text{ sec}) \times (60 \text{ HZ}) \\ &= 18000\end{aligned}$$

Thus,

$$\begin{aligned}\text{beginning count} &= 18000 - 1 \\ &= 17999\end{aligned}$$

However, this is a base 10 value. The corresponding hex value is 464FH. The timer #1 latch must be initialized with this value to create a 5-minute time delay using a 60 Hz clock.

22. Very long time delays are created using the PTM by cascading its timer sections together. Cascading the timers means that a given timer output clocks another timer.
23. The two continuous output modes available to the 6840 PTM are the 16-bit count mode, and the dual 8-bit count mode. For continuous output, the 16-bit count mode provides a 50% duty cycle, and the dual 8-bit count mode provides a variable duty cycle.

24. You must first calculate the output period, then take the reciprocal of the period to find the output frequency. With a 920 KHz internal clock frequency, and a beginning count value of 0355H the output period is:

$$\begin{aligned}
 \tau_o &= 2 \times \frac{(\text{beginning count} + 1)}{(\text{clock frequency})} \\
 &= 2 \times \frac{(0355 + 1)_{16}}{(920_{10} \text{ KHz})} \\
 &= 2 \times \frac{(0356_{16})}{(920,000_{10})} \\
 &= 2 \times \frac{(854_{10})}{(920,000_{10})} \\
 &= 2 \times .00093 \\
 &= .00186 \text{ sec.}
 \end{aligned}$$

Therefore,

$$\begin{aligned}
 f_o &= \frac{1}{\tau_o} \\
 &= \frac{1}{.00186 \text{ sec.}} \\
 &= 537.6 \text{ Hz}
 \end{aligned}$$



25. Using the dual 8-bit count mode, the output period is:

$$\begin{aligned}
 \tau_o &= \frac{(\text{MSB count} + 1) \times (\text{LSB count} + 1)}{\text{clock frequency}} \\
 &= \frac{(03_{16} + 1) \times (55_{16} + 1)}{920,000_{10}} \\
 &= \frac{(04_{16}) \times (56_{16})}{920,000_{10}} \\
 &= \frac{158_{16}}{920,000_{10}} \\
 &= \frac{344_{10}}{920,000_{10}} \\
 &= .00037 \text{ sec.} \\
 &= .37 \text{ milliseconds}
 \end{aligned}$$

Therefore,

$$\begin{aligned}
 f_o &= \frac{1}{\tau_o} \\
 &= \frac{1}{.37 \text{ milliseconds}} \\
 &= 2.70 \text{ kHz}
 \end{aligned}$$

26. The timer #2 output waveform duty cycle is:

$$\begin{aligned}
 \text{duty cycle} &= \frac{1}{(\text{MSB count} + 1)} \times 100\% \\
 &= \frac{1}{03_{16} + 1} \times 100\% \\
 &= \frac{1}{04_{16}} \\
 &= \frac{1}{4_{10}} \times 100\% \\
 &= 25\%
 \end{aligned}$$

27. Timer #3 must be configured in the one-shot, dual 8-bit mode to provide the desired output. The desired pulse width, or W, is 2 seconds. Recall that W equals the LSB count divided by the clock frequency, or:

$$W = \frac{\text{LSB count}}{\text{clock frequency}}$$

Therefore,

$$2 \text{ sec.} = \frac{\text{LSB count}}{1 \text{ Hz}}$$

Or,

$$\text{LSB count} = (2 \text{ sec.}) \times (1 \text{ Hz})$$

$$= 02_{10}$$

$$= 02_{16}$$

Now, the delay before the pulse must be 10 minutes, which is equivalent to 600 seconds. Recall that,

$$\text{Delay} = D = \frac{(\text{MSB count}) (\text{LSB count} + 1) + 1}{\text{clock frequency}}$$

Therefore,

$$600 \text{ sec.} = \frac{(\text{MSB count}) (02 + 1) + 1}{1 \text{ Hz}}$$

or,

$$(1 \text{ Hz}) \times (600 \text{ sec}) = (\text{MSB count}) (03) + 1$$

$$599_{10} = (\text{MSB count}) (03)$$

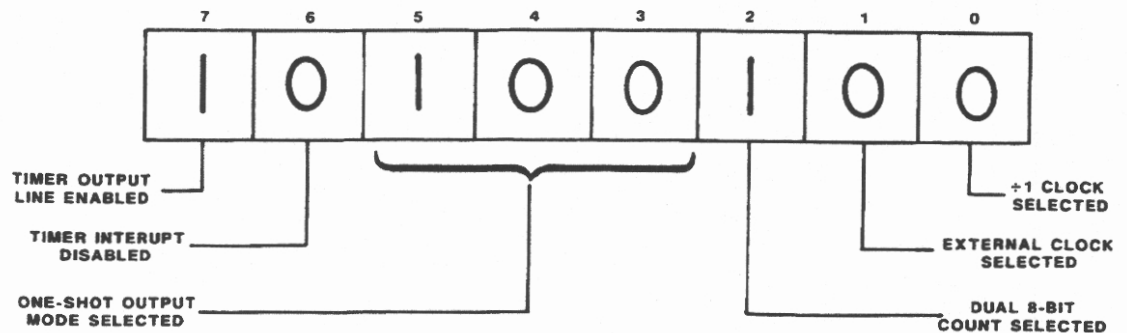
$$\text{MSB count} = \frac{599_{10}}{03_{10}}$$

$$= 199.66_{10}$$

$$= C7_{16}$$

Thus, the hex value that must be stored to the timer #3 latch is C702H.

28. To provide the output pulse desired in question 27, the timer #3 control register must be configured in the one-shot, dual 8-bit mode, with external clocking. The required control register configuration is shown in Figure 5-47. Thus, the hex value A4H must be stored to the timer #3 control register.



**Figure 5-47**

Timer #3 control register configuration for  
Self-Test Review question #28.

29. The following program will initialize the PTM to configure timer #2 in the period measurement mode, using internal clocking.

```

MVI A,4BH
OUT 5 IH
MVI A,00H
OUT 50H

```

30. The frequency of the unknown input signal is the reciprocal of its period. The period is:

$$\begin{aligned}\tau_{in} &= \frac{(\text{beginning count}) - (\text{final count})}{\text{clock frequency}} \\ &= \frac{FFFF_{16} - A105_{16}}{920_{10} \text{ kHz}} \\ &= \frac{5EFA_{16}}{920_{10} \text{ kHz}} \\ &= \frac{24,314_{10}}{920,000_{10}} \\ &= .0264 \text{ sec.}\end{aligned}$$

Therefore,

$$\begin{aligned}f_{in} &= \frac{1}{\tau_{in}} \\ &= \frac{1}{.0264 \text{ sec.}} \\ &= 37.87 \text{ Hz}\end{aligned}$$

## THE COUNTER/TIMER SECTION OF THE MUART

The 8256 MUART contains a counter/timer section that is capable of generating precise timed intervals and counting external events. The counter/timer section of the MUART consists of five separate 8-bit counter/timers that are basically programmable down counters. In other words, once a given timer is programmed with an initial value, it counts down to zero and times out. When the timer times out, an interrupt can be generated to the MPU.

### Counter versus Timer

The five counter/timers within the MUART are labeled Timer 1 through Timer 5. Timers 1, 4, and 5 can only operate as timers, while Timers 2 and 3 can operate as either timers or event counters. What's the difference? Well, as defined for the MUART, a timer is clocked from the internal MUART clock signal, while a counter is clocked from an external signal. Thus, the clocking source determines whether a given counter/timer is operated as a timer or an event counter.

Here's how it works. Suppose Timer 2 is configured to operate as a timer. By definition this means that Timer 2 will be clocked from the internal MUART clock signal. Now, recall that the internal MUART clock frequency is 1.024 MHz. This is too fast for many timing operations, so the MUART internally prescales this frequency to either 1 kHz or 16 kHz to drive the timer. You will see shortly how this option is software controlled. Suppose we select the 1 kHz prescale option and load Timer 2 with an initial value of, say 10H. Once Timer 2 is loaded with this value, it begins decrementing down to zero. When it reaches zero, an interrupt is generated via the MUART INT line. Now, how long will it take to reach zero? Well, the timer was initialized with the value 10H and is being clocked with a 1 kHz signal. Therefore, it will take  $10H \times 1 / 1 \text{ kHz}$ , or  $10 \times 1 \text{ ms} = 10 \text{ ms}$  to reach zero.

What is the shortest and longest time delay that can be achieved with a single timer? Well, using 16 kHz clocking, the shortest time delay would be  $01H \times 1 / 16 \text{ kHz}$ , or  $1 \times 62.5 \mu\text{s} = 62.5 \mu\text{s}$ . Using the 1 kHz clocking source, the longest delay would be  $FFH \times 1 / 1 \text{ kHz}$ , or  $255 \times 1 \text{ ms} = 255 \text{ ms}$ . Thus, a single timer can generate time delays that range from roughly 62.5  $\mu\text{s}$  to 255 ms, depending on the initial timer value and clocking source.

Now, suppose you could cascade two 8-bit timers together to form a single 16-bit timer. Then, you could extend the maximum timed interval to  $FFFFH \times 1 / 1$  kHz, or  $65,535 \times 1 \text{ ms} = 65,535 \text{ ms}$ . This is equivalent to 65.535 seconds. Consequently, a 16-bit timer could generate time delays that range from roughly 62.5  $\mu\text{s}$  to 65 seconds. You will see shortly that Timer 2 and Timer 4 can be cascaded together as well as Timer 3 and Timer 5 cascaded to form 16-bit timers within the MUART.

Now, how does an event counter differ from a timer? Earlier we stated that an event counter is clocked from an external source, rather than the internal MUART clock signal. Timer 2 and Timer 3 can be configured as event counters within the MUART. When configured to operate as an event counter, the value in the timer is decremented once for each low-to-high transition of an *externally* applied pulse as shown in Figure 5-48. Here you see that Timer 2 is being clocked from the signal being applied to the P12 port line of the MUART, while Timer 3 is being clocked via the P13 port line of the MUART. A value loaded into the timer is decremented with each pulse of the externally applied signal. Thus, the timer can be used to count the number of pulses applied. In addition, an interrupt can be generated to the MPU when the timer reaches zero.

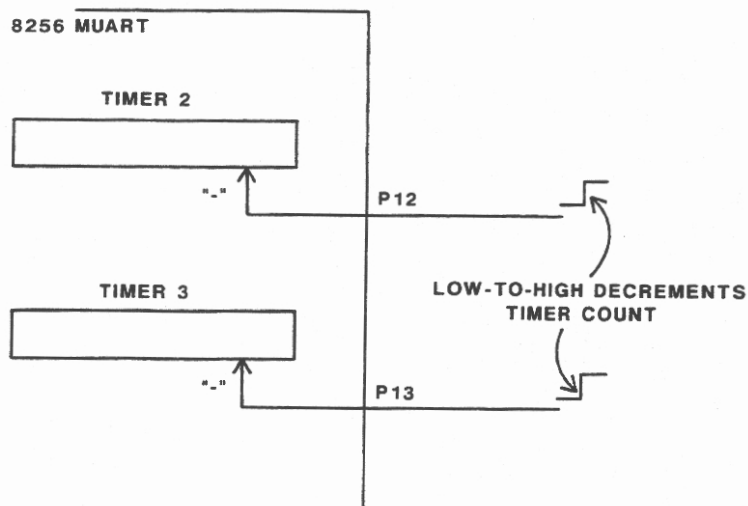


Figure 5-48

In the event counting mode, and *external* low-to-high pulse decrements the timer count.

You can also cascade Timer 2 with Timer 4 and Timer 3 with Timer 5 to form 16-bit event counters if desired. When this is done, the entire 16-bit value in the cascaded timers (2/4 or 3/5) is decremented with each low-to-high transition of the externally applied signal. Port line P12 is used to clock the Timer 2/4 combination, while port line P13 is used to clock the Timer 3/5 combination.

Recall that when a given timer reaches zero, an interrupt can be generated via the MUART INT line. How does the MPU know which timer generated the interrupt? Well, each timer is assigned a unique interrupt level as shown in Table 5-1.

Table 5-1  
Timer Interrupt Level Assignments

| Timer    | Interrupt Level | 8085 Restart Command | Address |
|----------|-----------------|----------------------|---------|
| 1        | Level 0         | RST0                 | 0H      |
| 2        | Level 1         | RST1                 | 4H      |
| 3 or 3/5 | Level 3         | RST3                 | CH      |
| 4 or 2/4 | Level 6         | RST6                 | 18H     |
| 5        | Level 7         | RST7                 | 1CH     |

Here you see the timer interrupt level as well as the associated 8085 Restart instruction and vector address. Thus, a Timer 1 interrupt will cause the 8085 MPU to vector to address 0H. Notice also that Level 3 and Level 6 interrupts are employed for the cascaded Timer 3/5 and Timer 2/4 operation, respectively. You should also be aware that some of these interrupt levels are also used for other interrupting sources within the MUART. For instance, the Level 7 interrupt is also used for Port 2 handshaking. The Mode Register configuration determines which interrupt source is active. In fact, the Mode Register is the key register when configuring the MUART timer section. With this in mind, let's learn more about how to configure the MUART for timer operation.

## Configuring the MUART Timer/Counters

The MUART Mode Register is shown in Figure 5-49. You were introduced to this register in Unit 3 when you configured Port 2 for parallel I/O. Recall that the lower three bits (P2C2 - P2C0) are used for this purpose. The upper five bits of the Mode Register are used to configure the MUART timers. These five bits are divided into three functional bit fields: the Counter/Timer Mode Control bits (CT3 and CT2), the Timer 5 Control bit (T5C), and the Cascade Timers bits (T35 and T24). A discussion of each functional bit field follows.

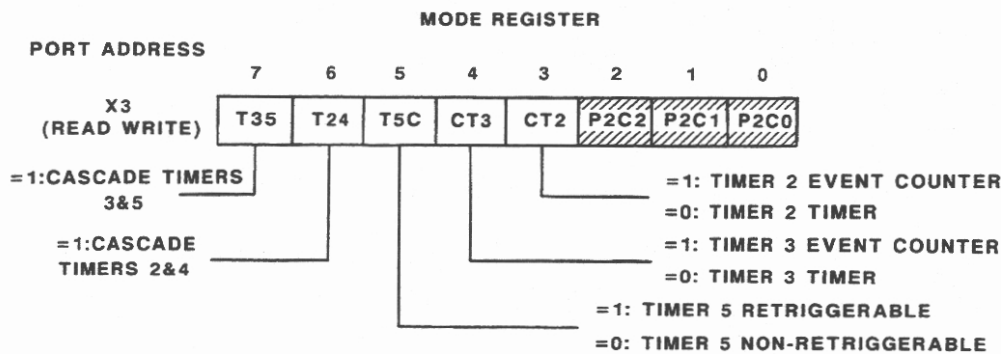


Figure 5-49

The Mode Register is used to configure the MUART for timer operation.

### COUNTER/TIMER MODE CONTROL (BITS CT3, CT2)

Recall that Timer 2 and Timer 3 can be used as event counters, even when they are cascaded with Timer 4 and Timer 5, respectively. To configure Timer 2 or Timer 3 as an event counter you must set its respective CT bit within the Mode Register. Thus, to configure Timer 2 as an event counter you must set CT2. When CT2 is set, Timer 2 is clocked from an externally applied signal at port line P12. If CT2 is cleared, Timer 2 is configured as a straight timer and clocked from the internal MUART clock signal (1 kHz or 16 kHz). Table 5-2 summarizes the function of the Counter/Timer Mode Control bits in the Mode Register.

Table 5-2  
Counter/Timer Mode Control

| CT3 | CT2 | Mode                                          |
|-----|-----|-----------------------------------------------|
| 0   | 0   | Timer 3 timer, Timer 2 timer.                 |
| 0   | 1   | Timer 3 timer, Timer 2 event counter.         |
| 1   | 0   | Timer 3 event counter, Timer 2 timer.         |
| 1   | 1   | Timer 3 event counter, Timer 2 event counter. |



## TIMER 5 CONTROL (BIT T5C)

This bit obviously controls the operation of Timer 5. Timer 5 is unique in the sense that it is a retriggerable timer. This means that it can be made to count over and over with a given value. In a normal count mode, Timer 5 is loaded with a value and counts down to zero just like any other timer in the MUART. In its retriggerable mode, Timer 5 is loaded with a value and doesn't begin to count until a high-to-low signal transition occurs on the P15 port line. Then, once counting, Timer 5 can be made to reload the initial value and recycle the count again by applying another high-to-low signal transition on the P15 port line.

When you load Timer 5 with an initial value, this value is internally saved within the MUART. When a high-to-low transition appears on the P15 port line, the value is loaded into Timer 5 and the timer begins counting down. This operation occurs each time a high-to-low signal transition occurs on the P15 port line.

What controls whether Timer 5 will operate in its normal or retriggerable mode? You're right, the T5C bit in the Mode Register. When the T5C bit is set, Timer 5 will operate in its retriggerable mode. On the other hand, Timer 5 will operate as a normal 8-bit timer when T5C is cleared.

Next, recall that Timer 5 can be cascaded with Timer 3 to form a 16-bit counter/timer. Timer 5 forms the most significant byte and Timer 3 the least significant byte of the 16-bit timer. This cascaded pair can be operated in the normal count mode or the retriggerable mode. When in the retriggerable mode (T5C = 1), the initial Timer 5 value is reloaded into the most significant byte each time a high-to-low transition occurs on P15. In addition, the least significant byte (Timer 3) is automatically set to all 1's (FFH). Thus an initial Timer 5 value of 55H would cause the Timer 3/5 pair to be reloaded with the value 55FFH each time a high-to-low transition occurred on the P15 port line.

## CASCADE TIMERS (BITS T35, T24)

These two bits in the Mode Register are used to cascade Timers 3 and 5, or Timers 2 and 4. Timers 2 and 3 always form the least significant byte, while Timers 4 and 5 form the most significant bytes of the respective pair. Timers 3 and 5 are cascaded when the T35 bit is set. Timers 2 and 4 are cascaded when the T24 bit is set within the Mode Register.

Table 5-3 summarizes the various counter/timer options controlled by the Mode Register. Review this table as a summary to the previous discussion.

Table 5-3  
MUART Timer Operation Summary

| Event Counter/<br>Timer | Function                                    | Programming<br>(Mode Word)   | Clock Source   |
|-------------------------|---------------------------------------------|------------------------------|----------------|
| 1                       | 8-Bit Timer                                 | —                            | Internal Clock |
| 2                       | 8-Bit Timer                                 | T24 = 0, CT2 = 0             | Internal Clock |
|                         | 8-Bit Event Counter                         | T24 = 0, CT2 = 1             | <u>P12</u>     |
| 3                       | 8-Bit Timer                                 | T35 = 0, CT3 = 0             | Internal Clock |
|                         | 8-Bit Event Counter                         | T35 = 0, CT3 = 1             | <u>P13</u>     |
| 4                       | 8-Bit Timer                                 | T24 = 0                      | Internal Clock |
| 5                       | 8-Bit Timer,<br>Normal Mode                 | T35 = 0, T5C = 0             | Internal Clock |
|                         | 8-Bit Timer,<br>Retriggerable Mode          | T35 = 0, T5C = 1             | Internal Clock |
| 2 and 4<br>Cascaded     | 16-Bit Timer                                | T24 = 1, CT2 = 0             | Internal Clock |
|                         | 16-Bit Event Counter                        | T24 = 1, CT2 = 1             | <u>P12</u>     |
| 3 and 5<br>Cascaded     | 16-Bit Timer,<br>Normal Mode                | T35 = 1, T5C = 0,<br>CT3 = 0 | Internal Clock |
|                         | 16-Bit Event Counter,<br>Normal Mode        | T35 = 1, T5C = 0,<br>CT3 = 1 | <u>P13</u>     |
|                         | 16-Bit Timer,<br>Retriggerable Mode         | T35 = 1, T5C = 1,<br>CT3 = 0 | Internal Clock |
|                         | 16-Bit Event Counter,<br>Retriggerable Mode | T35 = 1, T5C = 1,<br>CT3 = 1 | <u>P13</u>     |

## SELECTING THE INTERNAL TIMER CLOCKING SOURCE

There is one more control bit that you need to use when operating the MUART timers in their timer mode. This is the FRQ bit in Command Register 1. The FRQ bit is the least significant bit of Command Register 1 as shown in Figure 5-50. The FRQ bit selects between the two internal clocking frequencies available to all timers. Recall that when the timers are clocked internally they can be clocked from a 1 kHz (1 ms) or a 16 kHz (62.5  $\mu$ s) signal source. To select the 1 kHz clocking source, you must set the FRQ bit in Command Register 1. Conversely, the 16 kHz clocking source is selected when the FRQ bit is cleared. That's all there is to it!

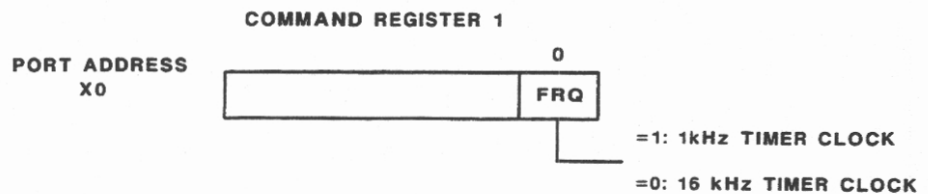


Figure 5-50

The FRQ bit within Command Register 1 selects between a 1 kHz or 16 kHz internal clocking source for the MUART timers.

## Self-Test Review

31. Explain the difference between a timer and an event counter as defined for the MUART.
32. What two internal frequencies are available for internal clocking for the MUART timers?
33. Suppose that Timer 2 is configured for timer operation and loaded with an initial value of 7FH. How long will it take Timer 2 to time-out using the 1 kHz internal clocking source?
34. What configuration and initial value are required for the MUART to generate a ten second time delay?
35. Which counter/timers within the MUART can be used as event counters?
36. Where must the external clocking signals be applied when the the timers in Question 35 are used as event counters?
37. Suppose Timer 2 is configured as an event counter? Where must the associated interrupt service routine jump instruction be located to service the Timer 2 interrupt?
38. What three functional bit fields within the Mode Register control the MUART timers?
39. Explain how Timer 5 is used as a retriggerable timer.
40. What bits within the Mode Register must be set to configure Timer 2/4 as an event counter and Timer 5 as a retriggerable timer?

## Answers

31. A timer is clocked internally and an event counter is clocked from an external signal.
32. 1 kHz and 16 kHz.
33.  $7FH \times 1 / 1 \text{ kHz} = 127 \times 1 \text{ ms} = 127 \text{ milliseconds}$
34. Timers 2 and 4 or Timers 3 and 5 must be cascaded and loaded with an initial decimal value of 10000, which is equivalent to the hex value 2710H.
35. Timer 2, Timer 3, Timer 2/4, or Timer 3/5.
36. For Timer 2 or Timer 2/4, the external pulse must be applied to the P12 port line. For Timer 3 or Timer 3/5, the external signal must be applied to the P13 port line.
37. Timer 2 is the source for a Level 1 interrupt which causes an RST1 instruction to be placed on the MPU data bus. The RST1 call location is 4H. Thus, a jump instruction to the Timer 2 interrupt service routine must be located at address 0004H.
38. Counter/Timer Mode Control (CT3,CT2).  
Timer 5 Control (T5C).  
Cascaded Timers (T35,T24).
39. Timer 5 can be used as a retriggerable timer by setting the T5C bit in the Mode Register. You must then initialize Timer 5 with a beginning count value. The timer will start counting down whenever a high-to-low signal transition is detected on the P15 port line. In addition, Timer 5 is automatically re-loaded with the same beginning count value and the count started over again with each subsequent high-to-low signal transition on the P15 port line.
40. The CT2 bit must be set to configure Timer 2/4 as an event counter.  
  
The T24 bit must be set to cascade Timers 2 and 4.  
  
The T5C bit must be set to configure Timer 5 as a retriggerable timer.

## UNIT SUMMARY

A device that can perform external timing operations is called a programmable timer. Programmable timers are used to free the MPU from such timing tasks as providing time delays, generating control pulses and waveforms, and measuring input pulses and waveforms. Once a timer is initialized by the MPU, additional service is not required by the MPU until the timer generates an interrupt. Thus, the timer provides the timing task and the MPU is free to perform other system tasks. A programmable timer is especially valuable in process control applications, where various pulse sensing and control tasks must be performed. Moreover, programmable timers are extremely useful in dedicated computer applications such as machine tools and automobiles.

The 6840 PTM is a typical programmable timer device. It contains three separate timer sections. Each section employs a counter, a latch, and a control register. The timer counters are 16-bit down counters which are read-only registers. An interrupt can be generated by the PTM each time a given counter times-out, or reaches 0000H. The timer latches are 16-bit write-only storage registers which are used to load the counter with a beginning count value. The control registers are 8-bit write-only registers that are used to control the operation of each respective timer section.

The output mode tasks of the PTM include interrupt generation, continuous waveform generation, and one-shot pulse generation. The input mode tasks of the PTM include interval measurement, pulse measurement, and period measurement. The respective timer control registers are used to configure a given timer for any of these output or input timing tasks.

The 8256 MUART also contains a timer section. The MUART timer section can perform simple time delays and count external events. The MUART timer section consists of five separate 8-bit counter/timer registers, called Timer 1 through Timer 5, that are basically programmable down counters. When operated as timers, these registers are loaded with a value and count down to zero at an internal clock frequency rate of either 1 kHz or 16 kHz. An interrupt can be generated to the MPU via the MUART INT line when a given timer reaches zero. When operated as event counters, Timer 2 and Timer 3 are loaded with a value then decremented once for each low-to-high transition of an externally applied signal. Thus, Timer 2 or Timer 3 can be used to count the number of external pulses applied. You can also cascade Timer 2 with Timer 4 and Timer 3 with Timer 5 to form a 16-bit timer or event counter. Finally, Timer 5 can be configured as a retriggerable timer whereby the timer is automatically loaded with the initial count value each time a high-to-low signal transition is detected on the P15 port line.

The MUART counter/timers are configured via the Mode Register. You configure Timer 2 or Timer 3 as either timers or event counters using the CT3 and CT2 bits within this register. Timers 3 and 5 or Timers 2 and 4 can be cascaded to form 16-bit counter/timers using the T35 and T24 bits of the Mode Register. Finally, Timer 5 is configured as a retriggerable timer using the T5C bit in the Mode Register.

When the MUART timers are clocked internally, the clocking frequency (1 kHz or 16 kHz) is selected with the FRQ bit in Command Register 1.

*Unit 6*

**ANALOG CONVERTER INTERFACING  
AND APPLICATIONS**



## CONTENTS

|                                                           |      |
|-----------------------------------------------------------|------|
| Introduction .....                                        | 6-3  |
| Unit Objectives .....                                     | 6-4  |
| Interfacing to D/A Converters .....                       | 6-5  |
| Self-Test Review .....                                    | 6-10 |
| Answers .....                                             | 6-11 |
| Microprocessor Applications Using D/A Converters .....    | 6-12 |
| Waveform Generation .....                                 | 6-13 |
| X-Y Displays .....                                        | 6-19 |
| Programmable Gain Amplifier and Attenuator .....          | 6-21 |
| Motor Control and Positioning .....                       | 6-23 |
| Process Control .....                                     | 6-29 |
| Analog Multiplexer .....                                  | 6-33 |
| Self-Test Review .....                                    | 6-36 |
| Answers .....                                             | 6-37 |
| Interfacing to A/D Converters .....                       | 6-39 |
| Interfacing and Controlling ADC Devices .....             | 6-39 |
| Handshake Control of an ADC .....                         | 6-40 |
| Interfacing and Controlling V/F Converters .....          | 6-45 |
| V/F Converter Interfacing Using Software Techniques ..... | 6-48 |
| Self-Test Review .....                                    | 6-51 |
| Answers .....                                             | 6-52 |
| Microprocessor Applications Using A/D Converters .....    | 6-54 |
| Instrumentation .....                                     | 6-54 |
| Data Acquisition .....                                    | 6-55 |
| Analog Multiplexers in a Data Acquisition System .....    | 6-56 |
| Sample/Hold Circuits in a Data Acquisition System .....   | 6-58 |
| Analog Sensing and Industrial Control Systems .....       | 6-63 |
| Self-Test Review .....                                    | 6-68 |
| Answers .....                                             | 6-69 |
| Unit Summary .....                                        | 6-70 |

## ***Unit 6***

# **ANALOG CONVERTER INTERFACING AND APPLICATIONS**

## **INTRODUCTION**

The world as we know it is an analog world. On the other hand, the microprocessor world is a digital world made up of 1's and 0's. Therefore, in order for a microprocessor to communicate with the real analog world, a digital-to-analog (D/A) or analog-to-digital (A/D) conversion must take place. A device called a DAC is used to provide D/A conversions, while ADCs and V/F converters are used to provide A/D conversions. In this unit, you will learn how to interface all of these devices to the MPU.

You will find both D/A and A/D converters in most all microprocessor-controlled industrial processes. The A/D converter is used to convert the output of a analog sensor to a digital value that can be evaluated by the MPU. The D/A converter is then used to generate analog control signals to the industrial process from digital control words received from the MPU. You will explore this and other applications of D/A and A/D converters in this unit.

## UNIT OBJECTIVES

1. Interface a D/A converter to a microprocessor system.
2. Write programs to generate waveforms from a microprocessor controlled D/A converter circuit.
3. Describe how a multiplying D/A converter circuit can be used to amplify or attenuate analog signals under control of a microprocessor.
4. Describe how D/A converters are used to control the direction of rotation, speed, and position of DC motors.
5. Define the function of a servo amplifier in a motor control circuit.
6. Interface parallel output A/D converters and V/F converters to a microprocessor via a MUART.
7. Describe how a MUART is used to provide handshake control of the analog conversion process of a parallel output A/D converter.
8. Write a program to count pulses generated by a V/F converter.
9. Describe the circuit requirements of a typical data acquisition system.
10. Describe how sample/hold, or S/H, devices and analog multiplexers are used with A/D converters in a data acquisition system.
11. Describe and provide an example of a microprocessor-based industrial control system.

## INTERFACING TO D/A CONVERTERS

Interfacing a 4, 6, or 8-bit digital-to-analog converter (DAC) to a microprocessor is relatively simple as illustrated in Figure 6-1. Since the DAC is an output device for the MPU, the DAC interface must consist of an address decoder and a latch.

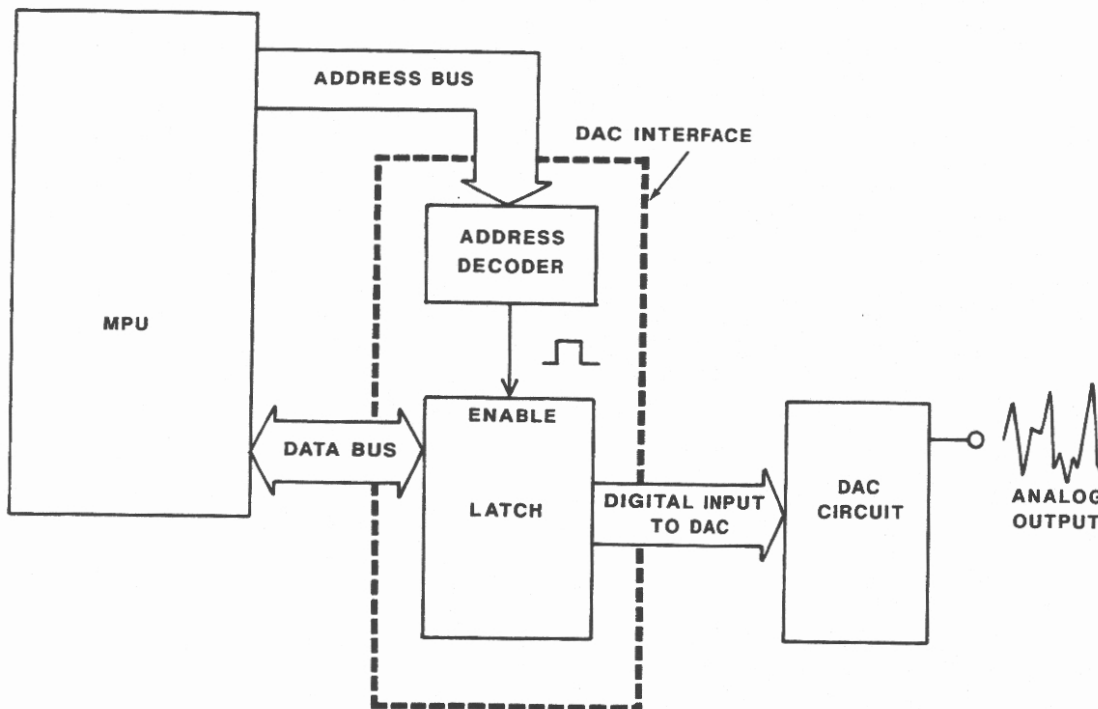


Figure 6-1

A DAC interface consists of an address decoder and a latch.

The DAC must first be assigned to an address within the memory address map of the system. When digital data is converted, the MPU writes, or stores, the data to the address assigned to the DAC. When this happens, the DAC address appears on the MPU address bus and, shortly thereafter, the data to be converted appears on the MPU data bus. Consequently, the address decoder must be designed to recognize the DAC address and ignore all other addresses on the address bus.

When the address decoder recognizes the DAC address, it enables the latch to capture the data on the data bus. The latch is required since the data to be converted is only present on the MPU data bus for a short time, typically less than .5 microseconds. The data to be converted is "held" on the latch output lines until new data is available. The DAC then makes the conversion and provides a proportional analog signal output. If you are interfacing an 8-bit DAC to the MPU, all eight data bus lines must be used. However, only four or six data lines are required for a 4-bit or 6-bit DAC interface, respectively.

I should mention that some DACs do not require the external latch shown in Figure 6-1. These DACs have their own internal latch, or buffer, and are sometimes referred to as **microprocessor compatible**. The Signetics NE5018 is a microprocessor compatible DAC. These devices have a line labeled "latch enable," or *LE*. The output of the address decode is connected directly to the DAC latch enable line.

As I stated earlier, it is more efficient and economical in many cases to use a MUART for the interface rather than combinational logic. Recall that the MUART has internal address decoding, latching, and three-state buffering. Thus, the DAC interface in Figure 6-1 can be replaced with a MUART as shown in Figure 6-2.

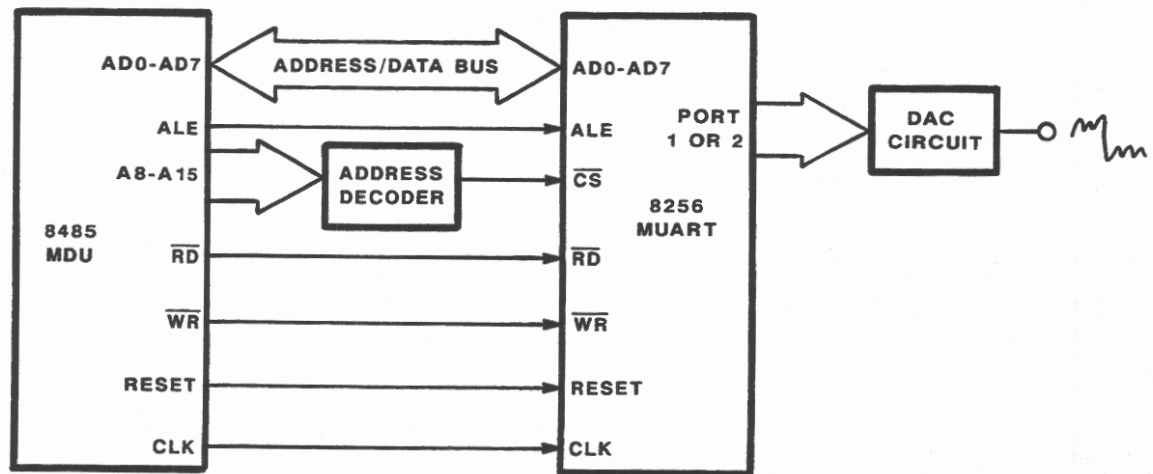


Figure 6-2  
The MUART used as a DAC interface.

Either Port 1 or Port 2 can be used to supply the digital value to the DAC for conversion. Of course, the port used must be configured for output when the MUART is initialized. An external latch is not required since output data is internally latched by the MUART ports.

Many applications require a greater resolution than can be provided by an 8-bit DAC. Fortunately, for these applications, there are 10, 12 and 14-bit DACs commercially available. These DACs may also be easily interfaced to your microprocessor. However, there is one small problem. When you connect a DAC of more than eight bits to the MUART, you must use both Port 1 and Port 2. When data is transferred to the MUART ports, both ports will not receive the data simultaneously. For example, suppose that you have connected a 10-bit DAC to the MUART such that the Port 1 lines provide the eight least significant DAC input lines and Port 2 lines 0 and 1 (*P20* and *P21*) provide the two most significant DAC input lines. Now, suppose the DAC is presently providing an analog signal which corresponds to the 10-bit digital value of 00 1000 0000. It is now desired to convert a new 10-bit value, say 10 0000 0000, which is clearly higher than the present value.

If you were to load this new value in the MPU index register, then store it to the MUART, what would happen? Since Port 1 forms the least significant eight bits of the 10-bit value, it would receive its eight new bits first, then the two most significant bits would be stored to Port 2. However, between the Port 1 and Port 2 operations, an intermediate and undesirable 10-bit data value would appear at the DAC input. This intermediate value would consist of the eight least significant bits of the new value from Port 1 and the two most significant bits of the old value from Port 2. With the two values we have chosen, the intermediate value would be 00 0000 0000. Obviously, this is undesirable. This problem is illustrated in Figure 6-3. Notice the "glitch" in the DAC output. The only solution to the problem is **double buffering**. That is, you must *latch* the *entire* 10-bit word before it is seen at the DAC input.

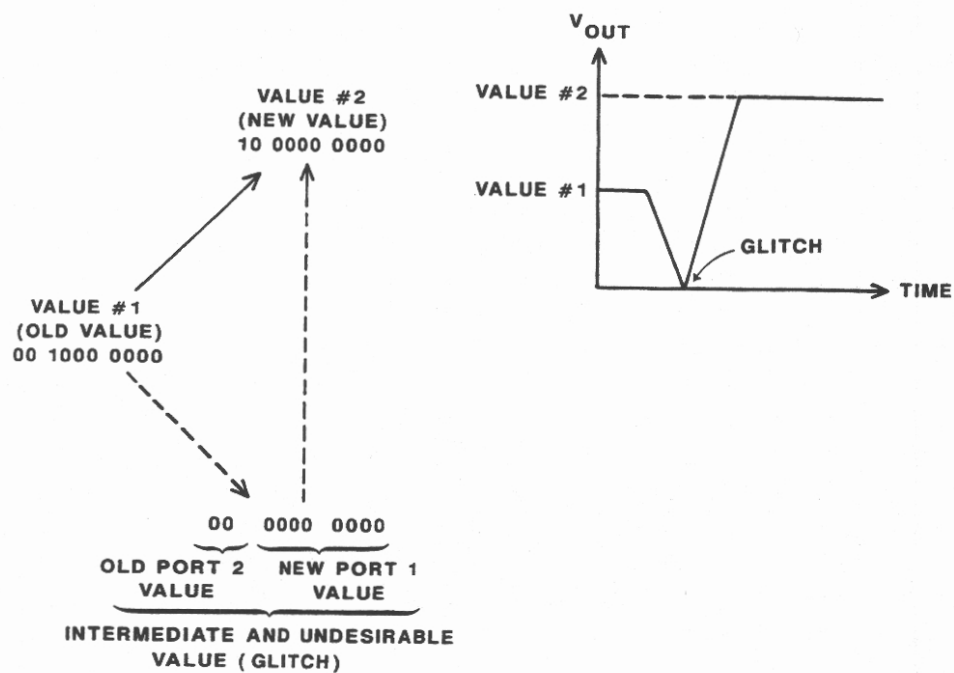


Figure 6-3

The "glitch" problem when interfacing DACs of more than 8-bits.

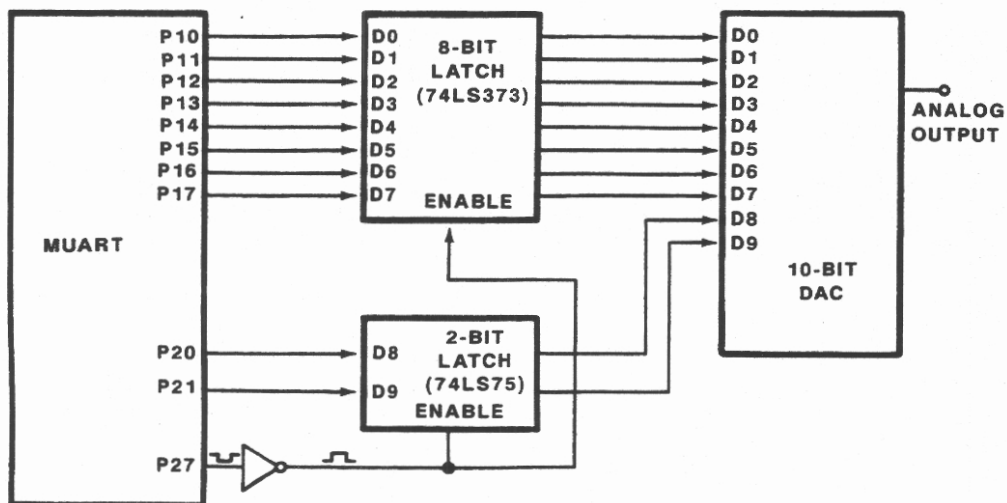


Figure 6-4

Double buffering for a 10-bit DAC using the MUART and external latching eliminates the glitch problem.

The circuit in Figure 6-4 shows one way that double buffering can be accomplished using the MUART and external latches. The 10-bit value is stored to Ports 1 and 2 from the MPU. Port 1 will receive the eight least significant bits first, then Port 2 will receive the two most significant bits.

Here, the MPU *P27* line is used to enable the latches. After the DAC data has been written to Port 1 and *P20*, *P21*, the MPU must toggle the *P27* line from low-to-high-to-low to enable the latches.

This results in the entire 10-bit value being passed to the DAC input lines simultaneously.

As stated earlier, some manufacturers advertise that their DACs are "microprocessor compatible." Recall that this means that the DAC has an internal latch. For less than eight bits, a single latch is provided. For more than eight bits, a double buffer, or latch, is provided internally. With these devices, no external latching is required.



## Self-Test Review

1. What does it mean to say that a DAC is "microprocessor compatible"?
2. Describe the interface for an 8-bit DAC.
3. What is required when interfacing a DAC of more than eight bits to an 8-bit microprocessor?
4. Explain why double buffering is required when interfacing with a DAC of more than 8 bits.
5. How could you connect a 12-bit unbuffered DAC to the MUART?

## Answers

1. A "microprocessor compatible" DAC usually has an internal latch, or buffer register.
2. An 8-bit DAC interface must consist of an address decoder and an 8-bit latch, unless the DAC is microprocessor compatible. A MUART can also be used as the interface between the MPU and DAC.
3. Double buffering must be provided between the microprocessor and the DAC input lines when the DAC is more than eight bits. If the DAC does not have any internal latching, double buffering can be provided with a MUART and an external latch(es).
4. Because the entire 12-bit value must be presented to the DAC at one time to prevent glitches in the DAC output for certain value transitions.
5. To connect a 12-bit DAC to the MUART, Port 1 could be used to supply the eight least significant input bits (*D0-D7*) to the DAC, and Port 2 used to supply the four most significant input bits (*D8-D11*) to the DAC. You must also provide a 12-bit latch between the MUART and DAC input lines. This can be done by using a 74LS373 8-bit latch and a 74LS75 4-bit latch. You must enable the latches by toggling one of the unused Port 2 output lines.

## MICROPROCESSOR APPLICATIONS USING D/A CONVERTERS

There are numerous microcomputer applications using digital-to-analog converters, especially in the control industry. As you will discover in this section, a microcomputer controlled DAC can be used to generate waveforms, control X-Y displays, control the speed and position of DC motors, and perform various process-related control tasks that require an analog control signal.

Before we get into specific DAC applications, we will discuss the role a DAC must play in a simple, closed-loop control system. In general, a microcomputer closed-loop control system requires two external analog functions as illustrated in Figure 6-5. First, the microcomputer must be capable of sensing an external condition. The sensing task is performed by a sensor, or transducer, that converts an analog quantity such as temperature into an analog current or voltage.

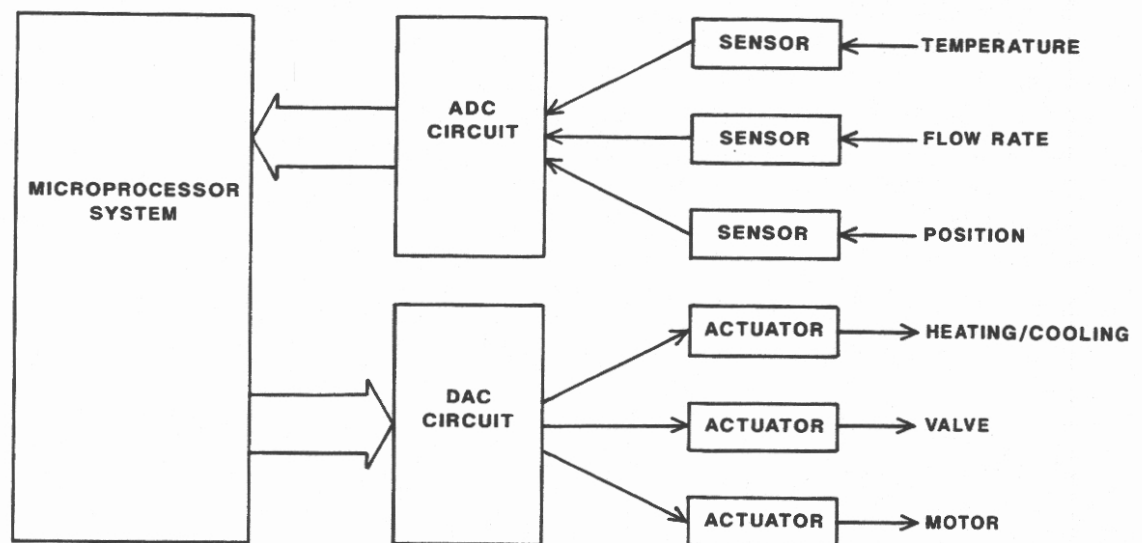


Figure 6-5  
A closed loop control system.

An analog-to-digital converter, or ADC, circuit converts the analog signal from the sensor into a digital word that can be read by the microcomputer. The microcomputer can then make a **control decision** based on the external condition. Once the control decision is made, the microcomputer must write a **control word** or series of control words to the DAC circuit. The DAC circuit then converts the control information to an analog signal which, in most cases, drives an actuator.

The actuator is a device that is used to control a motor, thermostat, valve, etc. The signal that the DAC circuit supplies to the actuator might be a steady-state level which is a function of the sensed condition, or continuous waveform whose amplitude and/or frequency is a function of the sensed condition.

In this section, we will show you how the DAC circuit is used to provide the various analog signals required for several common control tasks. Then, in subsequent units, you will learn about analog-to-digital converters, sensors, and actuators to complete the closed-loop control system.

## Waveform Generation

In many cases, a DAC circuit is required to generate a continuous waveform of a given amplitude and frequency. Depending on the application and control task, the waveform required could be a ramp (sawtooth), triangular, square, sine, or complex waveform. As you will soon discover, the waveform shape and frequency are controlled by software in the microprocessor. The waveform amplitude is controlled by the DAC circuit.

Many applications, like X-Y displays, recorders, and ADCs require the DAC circuit to generate a simple ramp, or sawtooth, waveform. If you interface the DAC circuit to the microprocessor with a MUART as shown in Figure 6-6, a simple program can be written to generate a ramp output. Here, we have connected an 8-bit DAC circuit to Port 2 of the MUART. However, Port 1 could also have been used in the same way. A flowchart for the ramp generation program is shown in Figure 6-7.

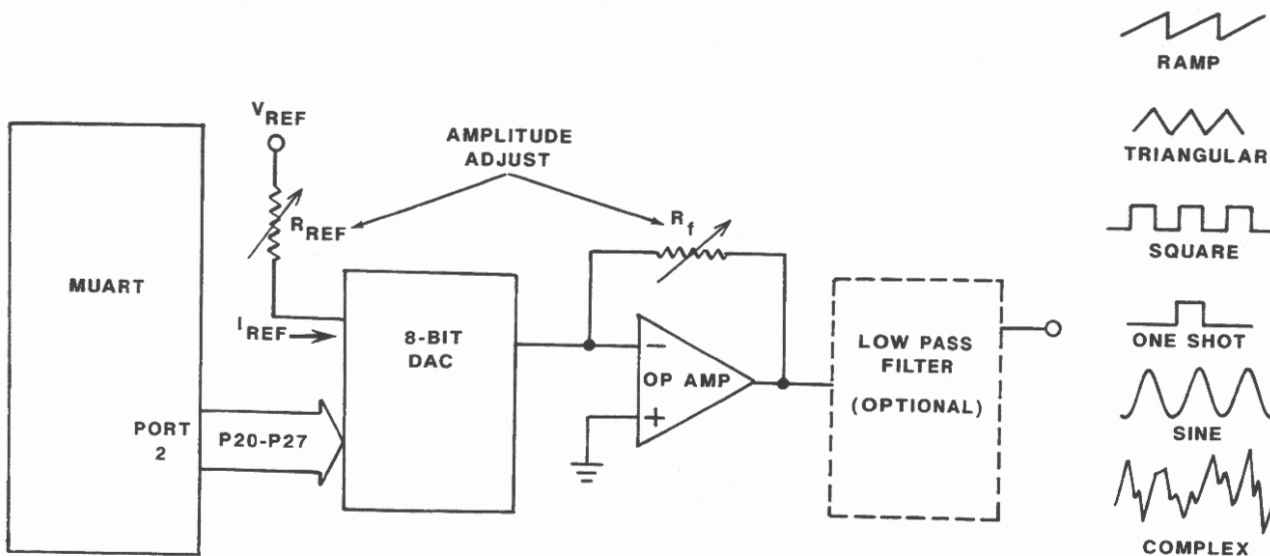


Figure 6-6

Waveform generation using a microprocessor controlled DAC circuit.

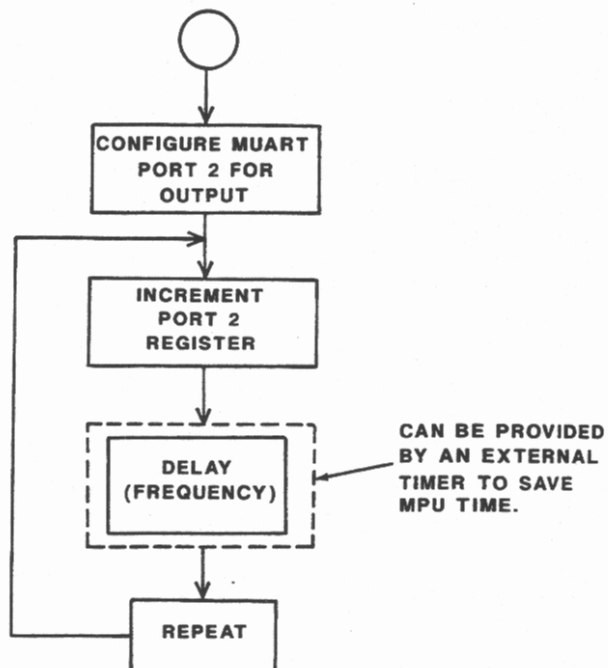


Figure 6-7

A program flowchart to generate a ramp, or sawtooth, output from the circuit in Figure 6-6.

The MUART is first initialized by configuring Port 2 for output. Then, Port 2 is repeatedly incremented to provide a continuous count to the DAC circuit. The output of a unipolar DAC circuit will cycle through its entire range and increase towards a maximum with each count, then drop to zero when the Port 2 count returns to zero, or rolls over. An inverted ramp output can be obtained from the DAC by using a decrementing routine instead of an incrementing routine.

The frequency of the output waveform is adjusted by inserting a delay between each increment (or decrement) step. The longer the delay, the lower the output waveform frequency. The required delay is created by incrementing or decrementing an internal MPU register. However, the MPU would not have any time to do anything else except generate the continuous output waveform if this were the case. Instead, many control circuits use an external timer to generate an interrupt to the MPU each time the output count is to be incremented. Thus, the time delay is performed by the external timer rather than the MPU. The MPU is then free to perform other tasks during the time delay period.

In addition, the application might only require that one or two output waveform cycles be generated at a given time. This further frees the MPU to perform other tasks between generating the required output cycles.

The amplitude of the output waveform can be adjusted with the DAC reference resistor,  $R_{REF}$ , or the op amp feedback resistor,  $R_f$ , in Figure 6-6. In addition, the output can be unipolar or bipolar, depending on how the DAC is connected.

Finally, recall that a DAC output is actually a staircase made up of individual steps. In most applications, this staircase is left unfiltered. However, you can filter the DAC output with a low pass filter if the staircase output is unacceptable. In general, the filter cutoff frequency must be equal to the highest frequency component (Fourier component) of the generated waveform.

A program flowchart to generate a triangular output waveform is shown in Figure 6-8. Notice that the triangular waveform program is a combination increment/decrement routine. Port 2 is incremented to a maximum value, then decremented back to a minimum value. The process is then repeated.

For a symmetrical output waveform, the two delays must be equal. Again, the delay periods control the output waveform frequency and are usually created by external timers to free the MPU for other tasks. The amplitude of the output waveform can be controlled by adjusting  $R_{REF}$  or  $R_f$  as before. In addition, the amplitude can be adjusted with software by changing the maximum and minimum count values.

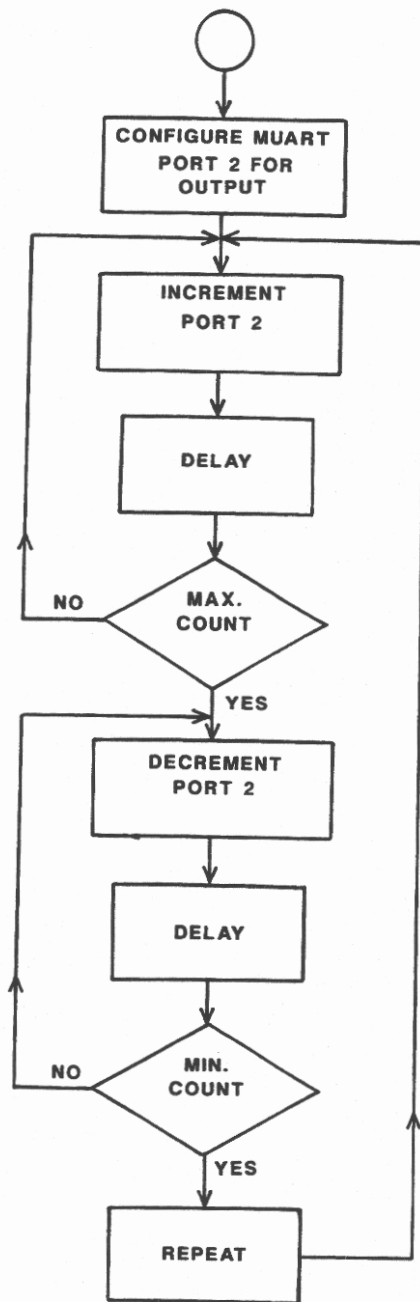


Figure 6-8  
A program flowchart to generate a triangular waveform  
from the circuit in Figure 6-6.

A program flowchart for generating square waveforms is shown in Figure 6-9. Here, after the MUART is initialized, a maximum value is stored to the Port 2 to obtain the maximum desired output level. After a delay, a minimum value is stored to Port 2 to obtain the minimum desired output level. Again, after a predetermined delay, the process is repeated to generate the continuous waveform.

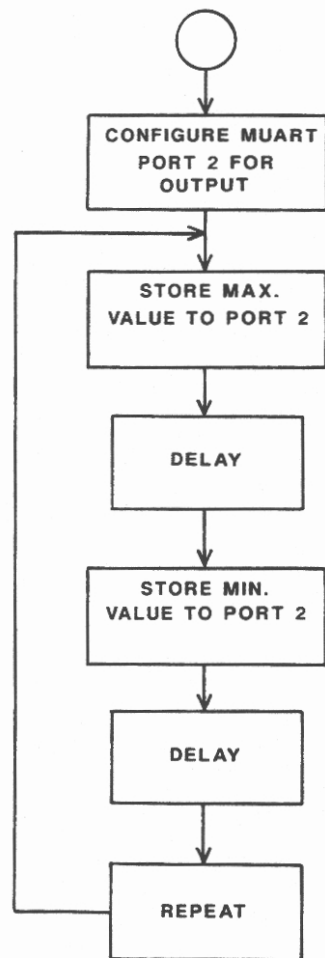


Figure 6-9

A program flowchart to generate a square waveform from the circuit in Figure 6-6.



As before, the delay periods control the frequency of the output waveform. In addition, the duty cycle of the output waveform can be adjusted by changing the delay periods. Equal delay periods provide a 50% duty cycle where the maximum time equals the minimum time, while unequal delay periods provide duty cycles other than 50%.

Again, the amplitude of the square waveform can be adjusted by changing  $R_{REF}$ ,  $R_F$ , or the maximum and minimum values stored to Port 2.

One-shot pulses can also be generated by the DAC circuit in Figure 6-6 by storing a value to Port 2 for a specific period of time. The one-shot program could be a subroutine that is called by the MPU whenever the one-shot pulse must be generated.

Finally, complex waveforms such as sine waveforms may be generated using more sophisticated software routines. In many cases, the software consists of a series of subroutines. The subroutines generate various time delays along with the standard ramp, triangular, and square waveform outputs. Complex waveform outputs can then be provided by linking these various subroutines together. The software required for complex waveform generation is beyond the scope of this course.

## X-Y Displays

Two DACs can be used as shown in Figure 6-10 to display a signal on an oscilloscope. One DAC is used to generate a ramp output to drive the X, or horizontal, trace of the oscilloscope. The frequency of the ramp waveform determines the sweep rate of the oscilloscope. A second DAC is then used to supply the signal to be displayed. The output of this DAC is applied to the Y, or vertical, trace of the oscilloscope.

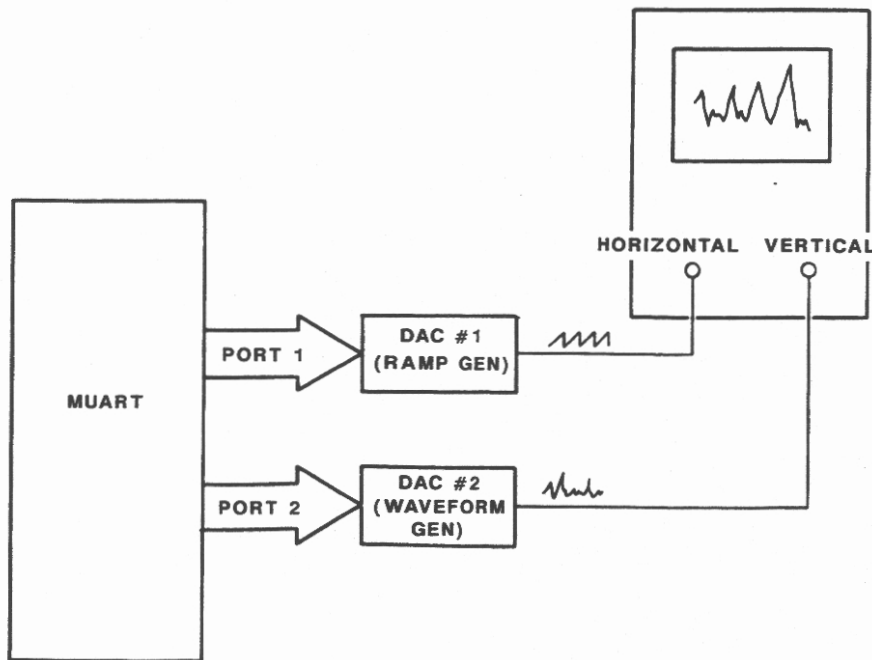


Figure 6-10

Two DACs being used to drive the X and Y inputs of an oscilloscope.

Other X-Y devices such as strip-chart recorders, graphic plotters, and coordinate axis machines require two bipolar DACs to control the X and Y coordinates required for positioning as illustrated in Figure 6-11. One DAC controls the +X or -X position and the other DAC controls the +Y or -Y position. A 3-axis machine would require three DACs, one to control each axis position, X, Y, and Z.

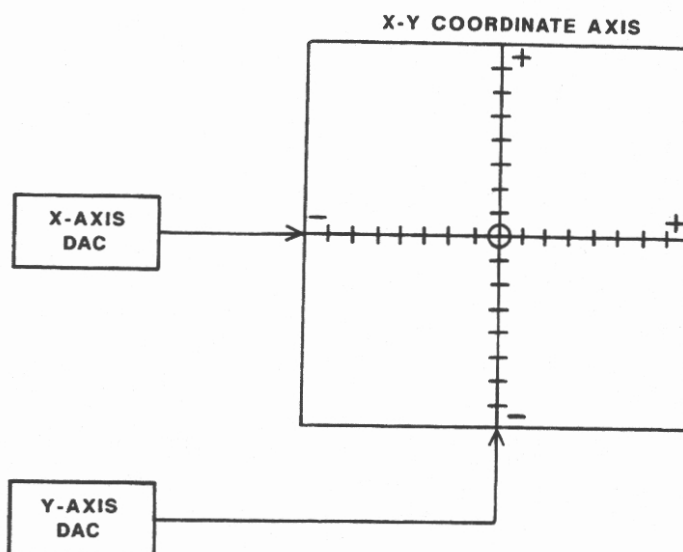


Figure 6-11

Two bipolar DACs can be used to position an object anywhere within an X-Y coordinate axis.

## Programmable Gain Amplifier and Attenuator

This application requires an 8-bit *multiplying* DAC. With a multiplying DAC, like the MC1408, the output voltage level is:

$$V_{OUT} = I_{REF} R_f \left[ \frac{D7}{2} + \frac{D6}{4} + \frac{D5}{8} + \frac{D4}{16} + \frac{D3}{32} + \frac{D2}{64} + \frac{D1}{128} + \frac{D0}{256} \right]$$

Where  $I_{REF}$  is the reference current applied to the DAC and  $R_f$  is the feedback resistor required for the current-to-voltage transducer op amp. However, recall that  $I_{REF} = \frac{V_{REF}}{R_{REF}}$ .

Consequently, the above equation can be rewritten as follows:

$$V_{OUT} = \frac{V_{REF}}{R_{REF}} R_f \left[ \frac{D7}{2} + \frac{D6}{4} + \frac{D5}{8} + \frac{D4}{16} + \frac{D3}{32} + \frac{D2}{64} + \frac{D1}{128} + \frac{D0}{256} \right]$$

Now, look at the circuit shown in Figure 6-12. If you apply an input signal to be amplified or attenuated to the  $V_{REF}$  input of the DAC circuit, the output voltage becomes:

$$V_{OUT} = V_{IN} \frac{R_f}{R_{REF}} \left[ \frac{D7}{2} + \frac{D6}{4} + \frac{D5}{8} + \frac{D4}{16} + \frac{D3}{32} + \frac{D2}{64} + \frac{D1}{128} + \frac{D0}{256} \right]$$

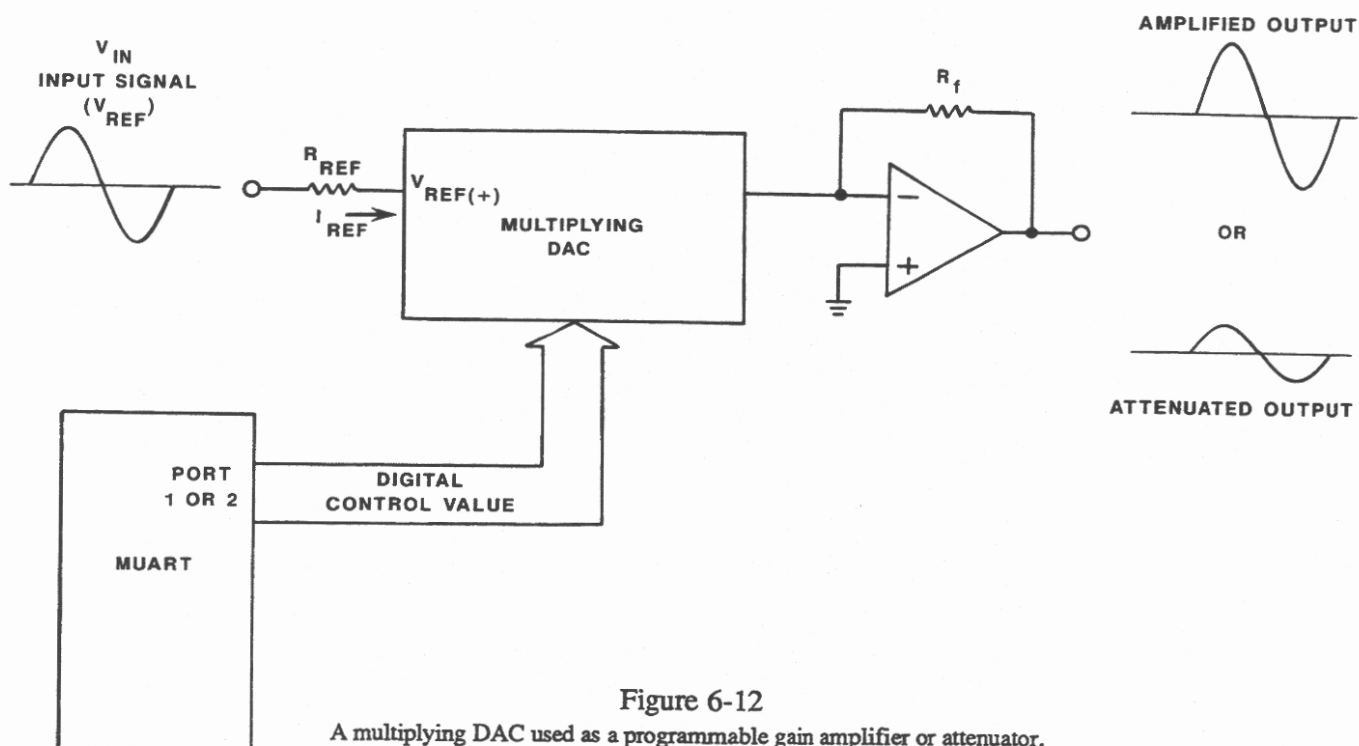


Figure 6-12

A multiplying DAC used as a programmable gain amplifier or attenuator. The digital value supplied to the DAC can be used to amplify or attenuate a signal input to the reference line of the DAC.

Where  $V_{IN} = V_{REF}$  = input signal to the DAC circuit. Next suppose you fix the value of  $R_{REF}$  and  $R_f$ . For example, suppose  $R_{REF} = 1 \text{ k}\Omega$  and  $R_f = 10 \text{ k}\Omega$ . Then, the output voltage becomes:

$$\begin{aligned} V_{OUT} &= V_{IN} \frac{10 \text{ k}\Omega}{1 \text{ k}\Omega} \left[ \frac{D7}{2} + \frac{D6}{4} + \frac{D5}{8} + \frac{D4}{16} + \frac{D3}{32} + \frac{D2}{64} + \frac{D1}{128} + \frac{D0}{256} \right] \\ &= V_{IN}(10) \left[ \frac{D7}{2} + \frac{D6}{4} + \frac{D5}{8} + \frac{D4}{16} + \frac{D3}{32} + \frac{D2}{64} + \frac{D1}{128} + \frac{D0}{256} \right] \end{aligned}$$

Notice that the input signal,  $V_{IN}$  is multiplied by ten times the digital input value. Consequently, the input voltage is increased (amplified) or decreased (attenuated) depending on the digital input value to the DAC. For example, suppose the digital input value to the DAC is  $FF_{16}$ , or  $1111 \ 1111_2$ . Then the output voltage,  $V_{OUT}$ , from the circuit in Figure 6-12 would be:

$$\begin{aligned} V_{OUT} &= V_{IN}(10) [.996] \\ &= 9.96 V_{IN} \end{aligned}$$

Thus, the input voltage is amplified by a factor of 9.96.

Next, suppose you change the digital input value to the DAC by writing the hex value  $15_{16}$  to the proper MUART port. Now the output voltage,  $V_{OUT}$ , becomes:

$$\begin{aligned} V_{OUT} &= V_{IN}(10) \left[ \frac{1}{16} + \frac{1}{64} + \frac{1}{256} \right] \\ &= V_{IN}(10)[.082] \\ &= .82 V_{IN} \end{aligned}$$

This time, the input voltage is attenuated by a factor of .82. You could say that the DAC is acting like a programmable potentiometer.

Thus, by applying a signal to the reference voltage input of a multiplying DAC, the signal can be amplified or attenuated. The DAC output will follow the input reference signal. By controlling the digital input value to the DAC with the MPU, you can control the amplitude of the output signal and, therefore, amplify or attenuate the input signal.

## Motor Control and Positioning

One of the most common uses of a DAC is for motor control and positioning. The DAC is required when the control or positioning operation is being performed by a digital circuit such as a microcomputer.

The direction and speed of a DC motor can be controlled with the analog output of a DAC as illustrated in Figure 6-13. The direction of rotation of the motor is controlled by the polarity of its control voltage. The speed of a DC motor is controlled by the amount of control voltage applied to the motor. In Figure 6-13, a bipolar DAC is used to allow variable speed control in both directions of rotation. One output polarity will cause the motor to rotate in a given direction. The opposite output polarity will reverse the direction of rotation. The speed of rotation is controlled, or adjusted, by the digital value written to the DAC via the MUART interface discussed earlier.

Even small DC motors require relatively high current levels for operation. The output of the DAC circuit cannot supply the current required by the motor. Consequently, a **servo amplifier** is used in Figure 6-13 between the DAC and the motor. A servo amplifier is nothing more than a power op amp that is capable of supplying the current levels required by the motor.

The motor control circuit shown in Figure 6-13 is an *open-loop* circuit since no feed back signal is provided to "tell" the microprocessor the position or speed of rotation of the motor shaft.

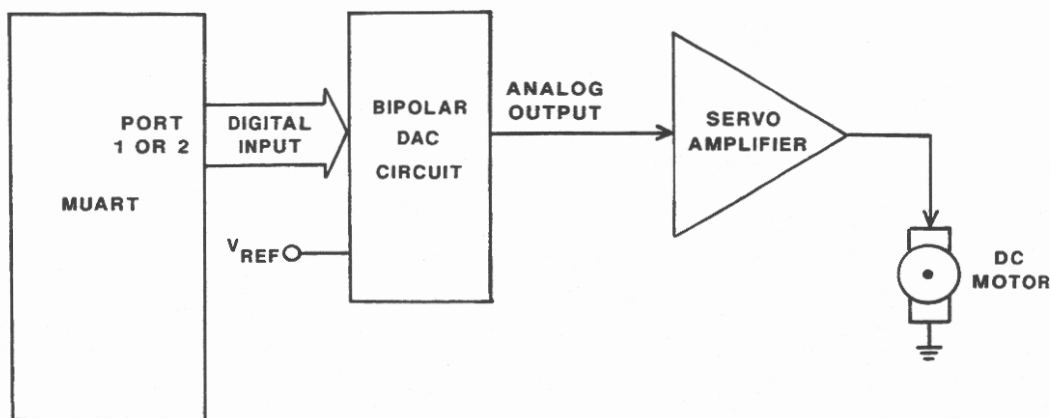


Figure 6-13

An open loop motor control circuit using a microprocessor control DAC circuit.

Many times, a DC motor is used to control the position of an object. When this is the case, a *closed-loop* control circuit such as the one shown in Figure 6-14 must be used. Here, a **feedback potentiometer** and **comparator** have been added to detect the relative position of the motor shaft. In addition, a unipolar DAC circuit is used. Here's how it works. The wiper of a linear potentiometer is ganged, or geared, to the shaft of the motor. In addition, the potentiometer is connected between ground and the  $V_{REF}$  supply for the DAC. Consequently, as the motor shaft turns, the voltage at the wiper of the potentiometer varies between 0 V and  $V_{REF}$ . The potentiometer voltage is applied to one input line of a comparator. The other comparator input line is connected to the analog output signal from the unipolar DAC circuit.

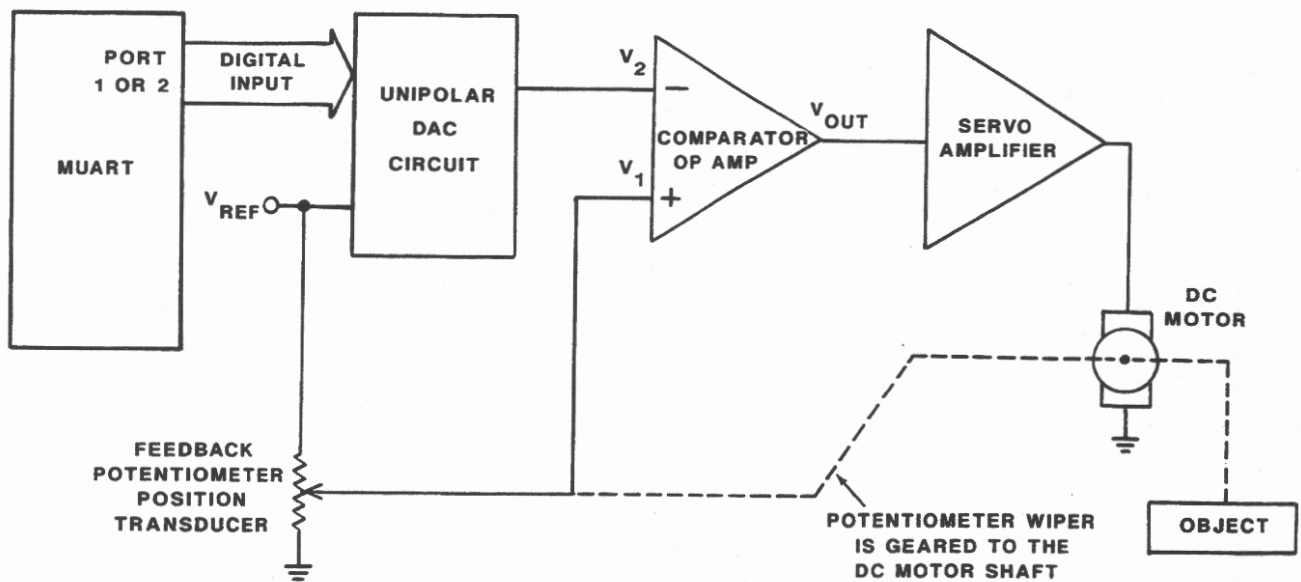


Figure 6-14

A closed loop circuit for constant speed control of position.

Before we go any farther, we should review the operation of comparators since they are very common in control circuits. A **comparator** is a circuit that compares two voltages. The output from the comparator indicates which input voltage is larger. Op amps are often used as comparators, as shown in Figure 6-15. Here, the two input voltages,  $V_1$  and  $V_2$ , are to be compared. Any difference between  $V_1$  and  $V_2$  is amplified by the open-loop gain of the op amp. The open-loop gain of standard op amps typically ranges from 10,000 to 250,000. Consequently, any difference between  $V_1$  and  $V_2$  causes the output of the op amp to saturate in one direction or the other as shown in Figure 6-15. Notice that  $V_1$  is connected to the non-inverting input of the op amp and  $V_2$  is connected to the inverting input of the op amp. If  $V_1$  is greater than  $V_2$ , the output voltage goes to its maximum positive value, or  $+V_{SAT}$ . Conversely, if  $V_1$  is less than  $V_2$ , the output voltage goes to its maximum negative value, or  $-V_{SAT}$ . Both saturation levels,  $+V_{SAT}$  and  $-V_{SAT}$ , are typically one or two volts less than the plus and minus supply voltages,  $V_+$  and  $V_-$ . You will observe the use of op amp comparators in many of the control circuits that follow.

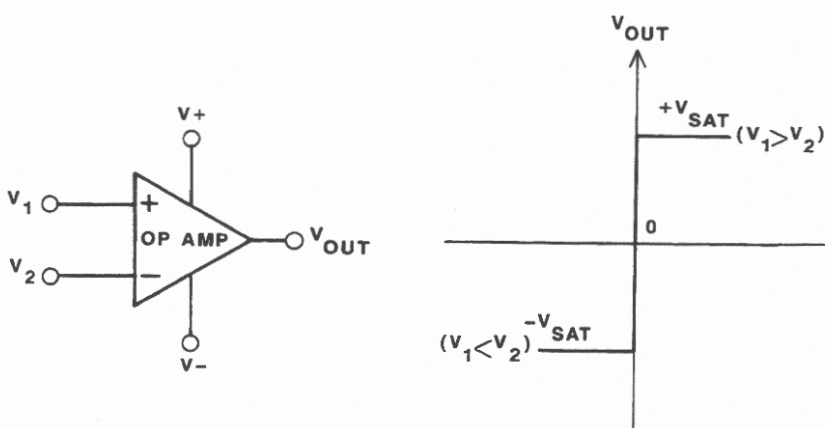


Figure 6-15

A comparator is an op amp circuit that is used to compare two voltages. The output from the comparator indicates which input voltage is larger.

Now, back to the closed-loop control circuit in Figure 6-14. The motor shaft position signal provided by the potentiometer is connected to the non-inverting input of the comparator. The analog output of the DAC circuit is connected to the inverting input of the comparator. The microcomputer must first determine the desired position of the object and write the 8-bit binary equivalent to the MUART.



Depending on the application and how the motor is geared, there will be a certain range of motion for the position of the object being controlled. A binary value of 0000 0000 will represent one end of the position range and a value of 1111 1111 is used to represent the opposite end of the position range. The binary equivalent of the desired object position is written to the MUART and converted to a proportional analog voltage by the DAC circuit. This voltage is applied to the inverting input of the comparator. Thus, the comparator "sees" the new position voltage,  $V_2$ , at its inverting input provided by the DAC circuit and the old position voltage,  $V_1$ , at its non-inverting input provided by the potentiometer. Now, three possibilities exist:

1. The new position voltage is the same as the old position voltage, or  $V_2 = V_1$ .
2. The new position voltage is **higher** than the old position voltage, or  $V_2 > V_1$ .
3. The new position voltage is **lower** than the old position voltage, or  $V_2 < V_1$ .

When the new position and old position voltages are the same, the comparator output,  $V_{OUT}$ , is zero and the motor shaft does not turn.

When the new position voltage is higher than the old position voltage (Case 2), the comparator output voltage is negative and the object will move in a given direction until the new desired position is reached. Notice that as the object moves, the voltage value from the potentiometer tracks the object movement as it approaches the desired position. When the object reaches the desired position, the DAC and potentiometer voltages are equal. Consequently, the comparator output,  $V_{OUT}$ , drops to zero and the motor shuts off. Furthermore, the comparator will output a constant voltage level until the desired position is reached. Therefore, the motor shaft will turn at a constant speed and the object will move at a constant rate.

Finally, if the new position voltage is less than the old position voltage (Case 3), the comparator output is positive and the object moves in the opposite direction until the new desired position is reached. Again, the object will move at a constant rate until the new position is reached. At this point, the output of the comparator falls back to zero and the motor shuts off.

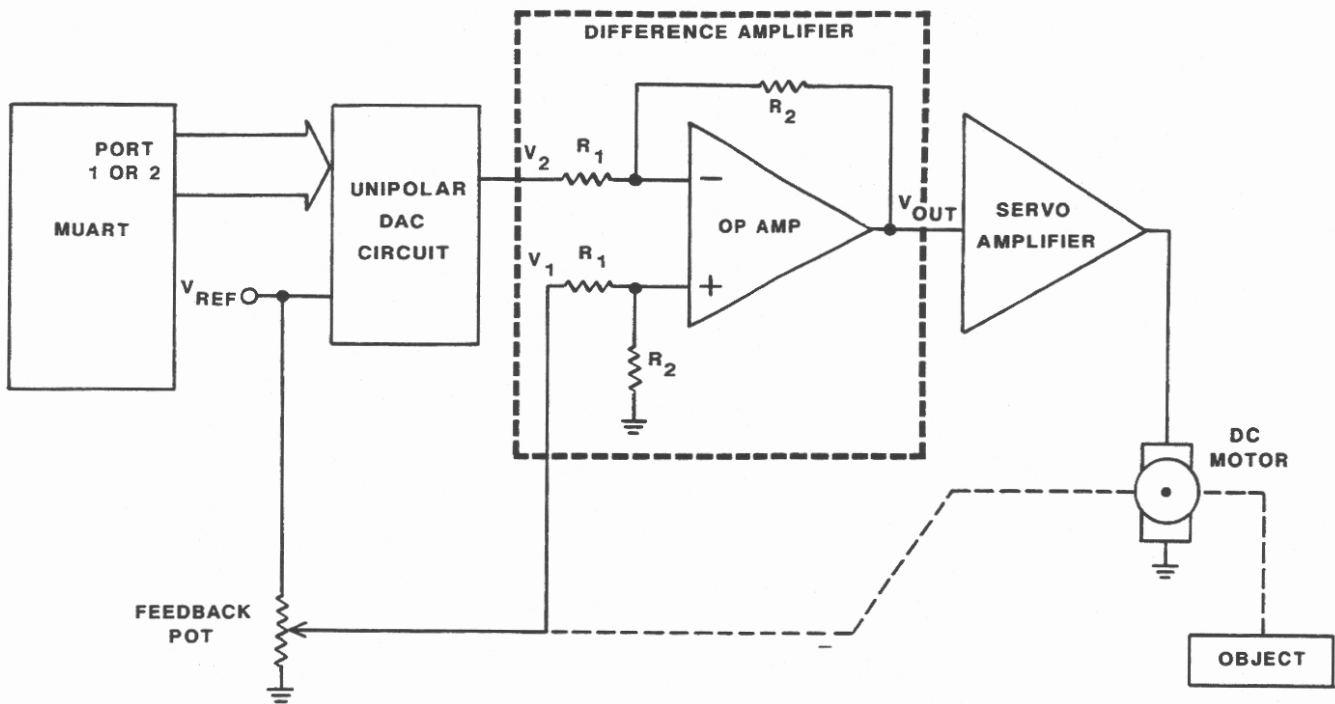


Figure 6-16

A closed loop circuit for variable speed control of position.

The closed-loop control circuit just discussed provides for constant movement of an object to the desired position. Sometimes it is desirable to move the object at a variable speed so that when the object is far away from the desired position, it will move faster, then gradually slow down until the position is reached. This can be accomplished by using a difference amplifier in place of a comparator as shown in Figure 6-16. A difference amplifier is another configuration of an op amp. With this configuration, the output voltage can be calculated as follows:

$$V_{OUT} = \frac{R_2}{R_1} (V_1 - V_2)$$

Notice from the above equation that, since  $R_1$  and  $R_2$  are constant, the differential amplifier output voltage,  $V_{OUT}$ , level is a direct function of the difference between the two input levels,  $V_1$  and  $V_2$ . If  $V_1$  and  $V_2$  are equal,  $V_{OUT}$  is zero and the object will not move. If  $V_2 < V_1$ ,  $V_{OUT}$  is positive, and the object will move in one direction. Conversely, if  $V_2 > V_1$ ,  $V_{OUT}$  is negative, and the object will move in the opposite direction.

However, unlike the previous circuit,  $V_{OUT}$  is not constant. A large difference between  $V_1$  and  $V_2$  causes a higher  $V_{OUT}$  level than does a small difference between  $V_1$  and  $V_2$ . Consequently, the object begins moving fast then gradually slows down until it reaches the desired position.

Another method for controlling the position of an object is illustrated in Figure 6-17. This circuit uses a **digital position encoder** to "tell" the microprocessor the position of the object. The digital position encoder is connected to the motor shaft. Actually, the digital position encoder is an analog-to-digital converter, or ADC. The output of the encoder is a binary or BCD value that is input to the MUART. We have chosen to use Port 2 of the MUART in Figure 6-17.

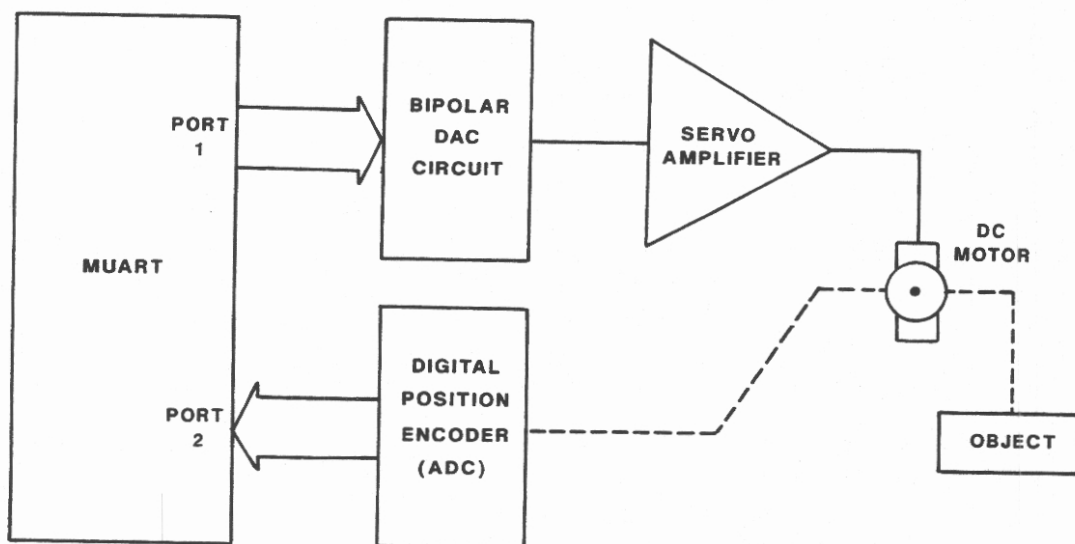


Figure 6-17

A closed loop control circuit using a digital position encoder.

To move the object to a new position, the microprocessor turns on the DAC by writing the new position value to Port 1 of the MUART. The microprocessor then continuously reads Port 2 of the MUART and compares the actual position value supplied by the encoder to the new position value which was written to the DAC at Port 1. When the two values are equal, the microprocessor turns the motor off by writing a value to the DAC which provides a zero output to the motor. Notice that a bipolar DAC must be used in this circuit to allow object movement in both directions.

The disadvantages of the circuit shown in Figure 6-17 should be obvious. First, the digital position encoder is much more complex and probably more expensive than the potentiometer/op amp circuits in the previous two circuits. However, in most cases it will provide a more accurate and linear indication of the object position.

Another disadvantage of using digital position encoder circuit is that the comparison is made by the microprocessor software rather than hardware. This may or may not be desirable, depending on the system demands on the MPU. If the MPU's time is critical, it is better to perform the comparison with external hardware as shown in Figures 6-14 and 6-16. This frees the MPU to perform more important system related tasks.

Finally, the position resolution of all the previous closed-loop control circuits is a function of the DAC size. A 6-bit DAC will provide 64 position locations, while an 8-bit DAC will provide 256 position locations.

## Process Control

Many times, the analog output of a DAC or several DACs is used to control a production or testing process. A microprocessor that is controlling a production or testing process will provide sequential digital control signals, to control various actuating devices such as valves, motors, relays, and thermostats. Consequently, several actuating devices are being controlled by a single microprocessor control system. An example is a heat treating process where a product is exposed to a heating profile as it passes through a belt furnace. The furnace is internally divided into several temperature zones. Each temperature zone must be separately controlled to provide the overall temperature profile required as the product moves through the furnace. The analog output of a DAC can be applied to a temperature controller to provide the required temperature for a given zone.

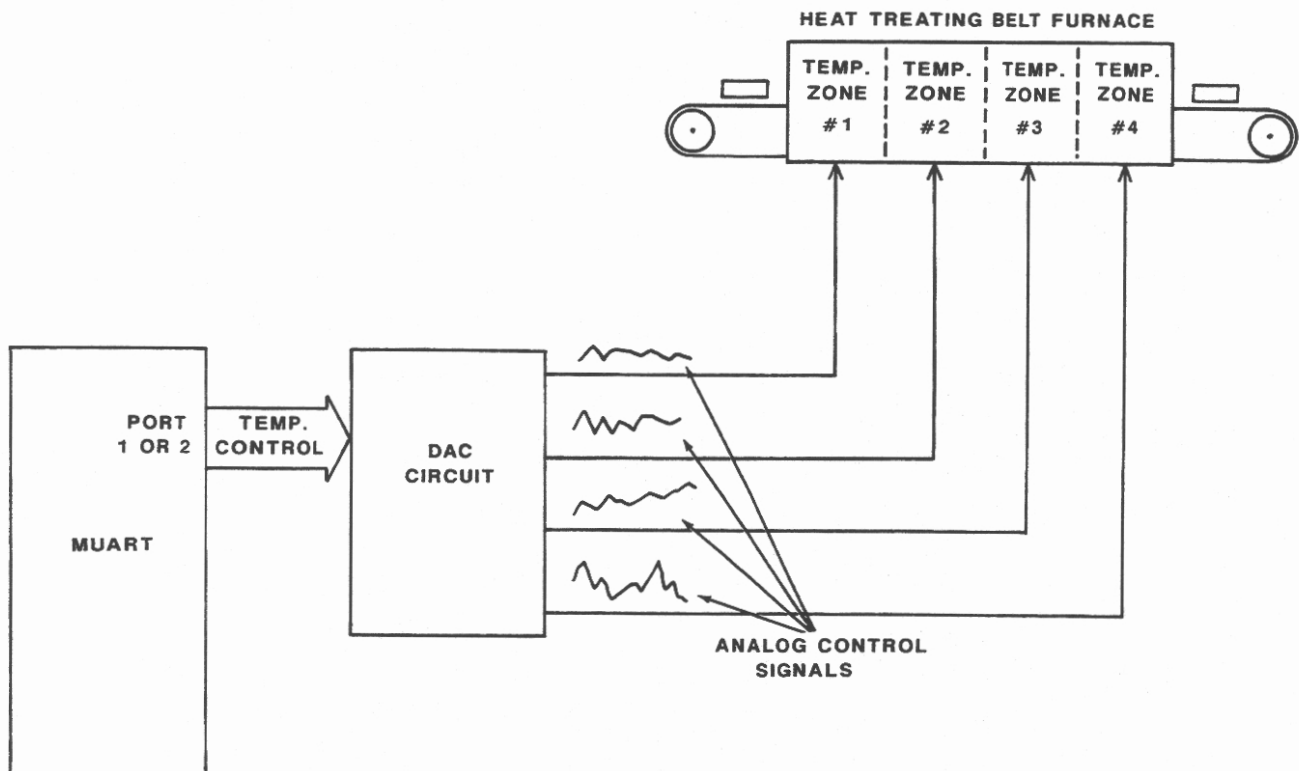


Figure 6-18  
A microprocessor system and a DAC used for control of  
a heat treating process.

You might think that a single DAC can be used between the microprocessor system and furnace as shown in Figure 6-18 to provide the required temperature control. However, you will note that there is a problem with the diagram in Figure 6-18. A DAC has only one output line, and there are several zones to be controlled. How can this single output line control several different control zones? The obvious solution to the problem is to use several DACs, one for each control zone, as shown in Figure 6-19. Here, two MUARTs are used to supply the temperature control information to four 8-bit DACs. A given temperature zone control word is written to a given DAC via its respective MUART port. Each DAC output is supplied to its associated temperature controller in the furnace and remains constant until a new zone control word is provided by the microprocessor. However, as you can see, this solution requires several additional components that are relatively expensive. More control zones would require even more MUARTs and DACs.

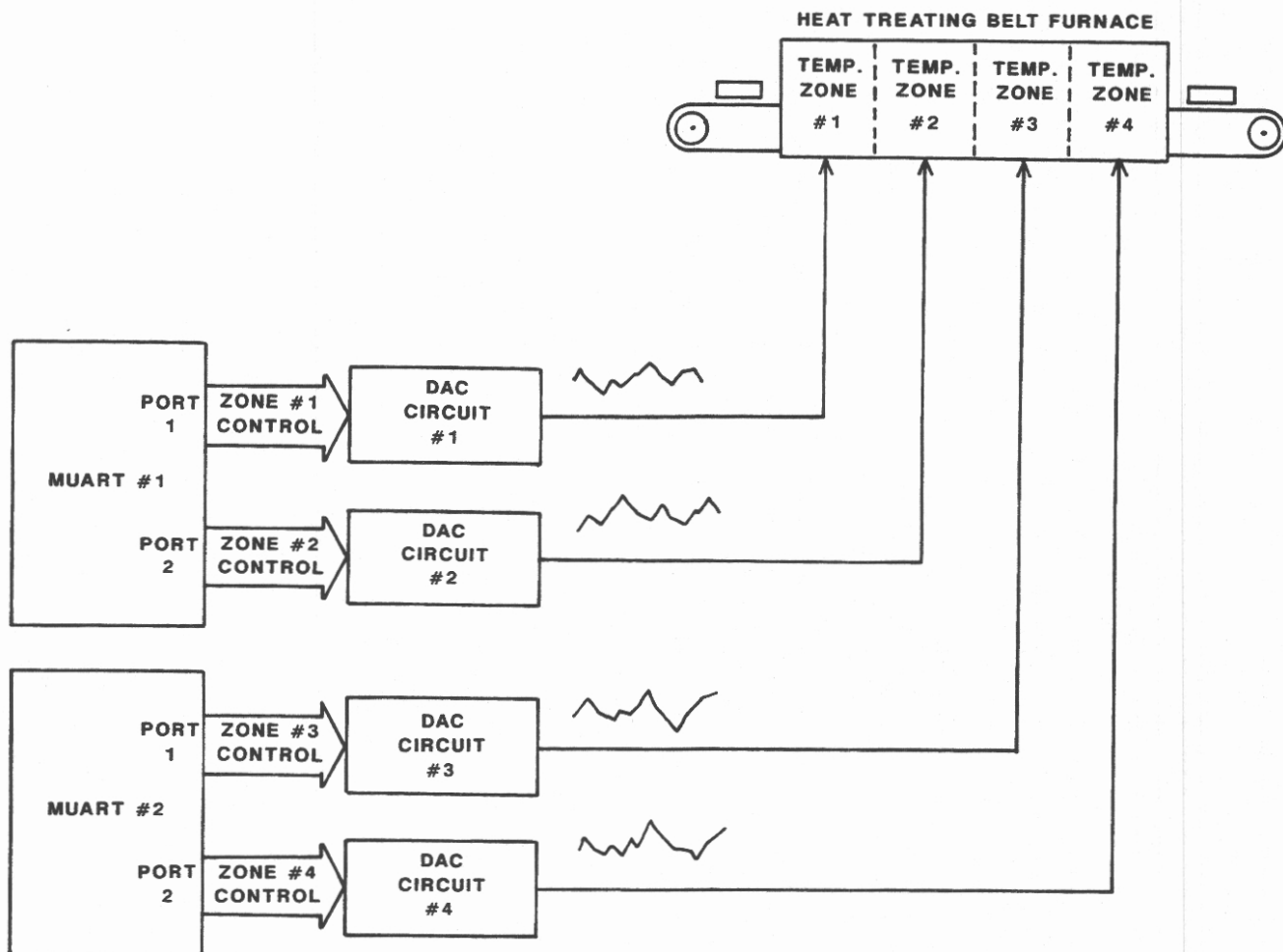


Figure 6-19

Several DACs can be used for control of a process - one for each control zone.

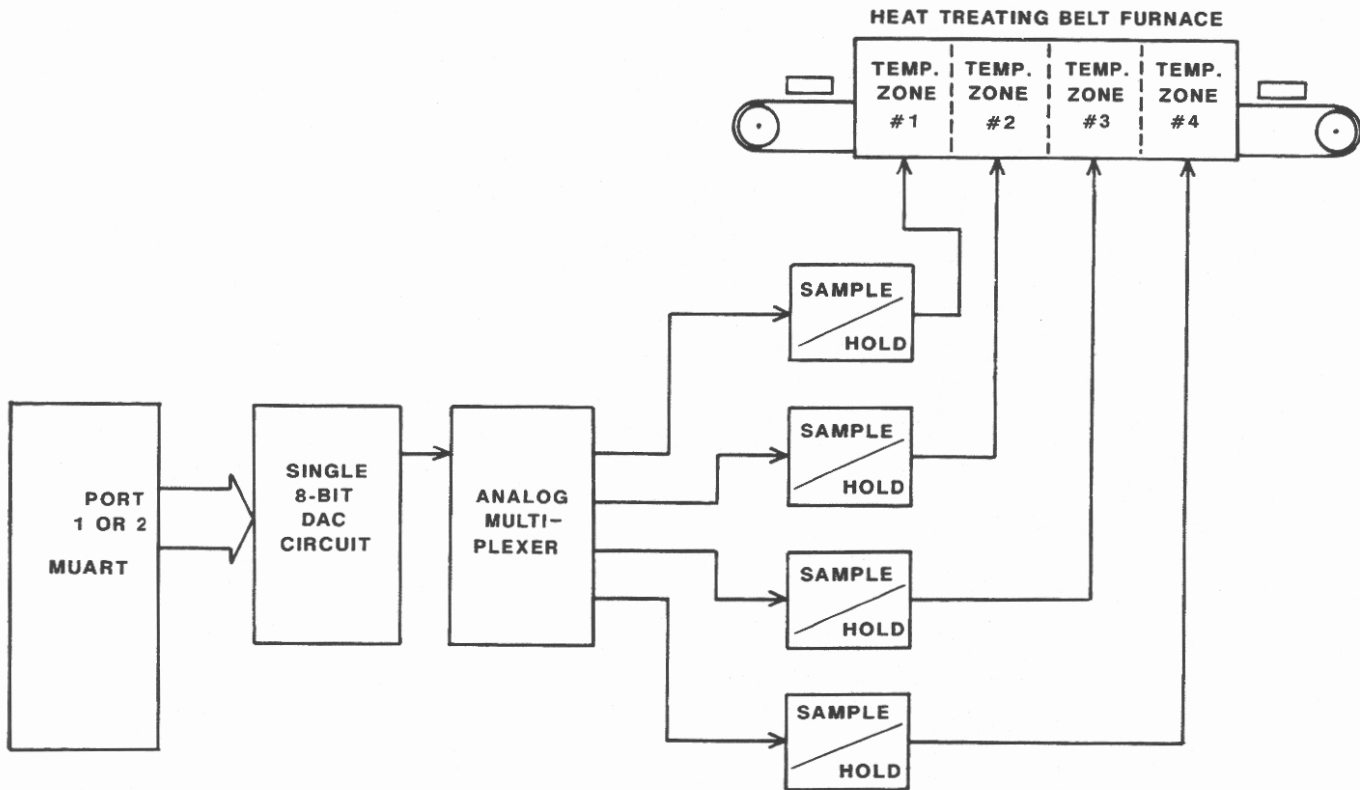


Figure 6-20

The output of a single DAC can be multiplexed and held to supply several separate control signals.

An alternative to the multiple DAC solution is to use a single DAC and multiplex and hold its analog output signal as shown in Figure 6-20. A common analog multiplexing device and one sample/hold is required for each output control signal. The DAC performs the conversions for all four temperature control zones, and the analog multiplexer applies the output of the DAC to the appropriate sample/hold circuit. The analog multiplexer is a series of analog switches, and the sample/hold circuit is a voltage memory, or storage, device. An actual multiplexing circuit is shown in Figure 6-21. Let's take a closer look at this circuit.

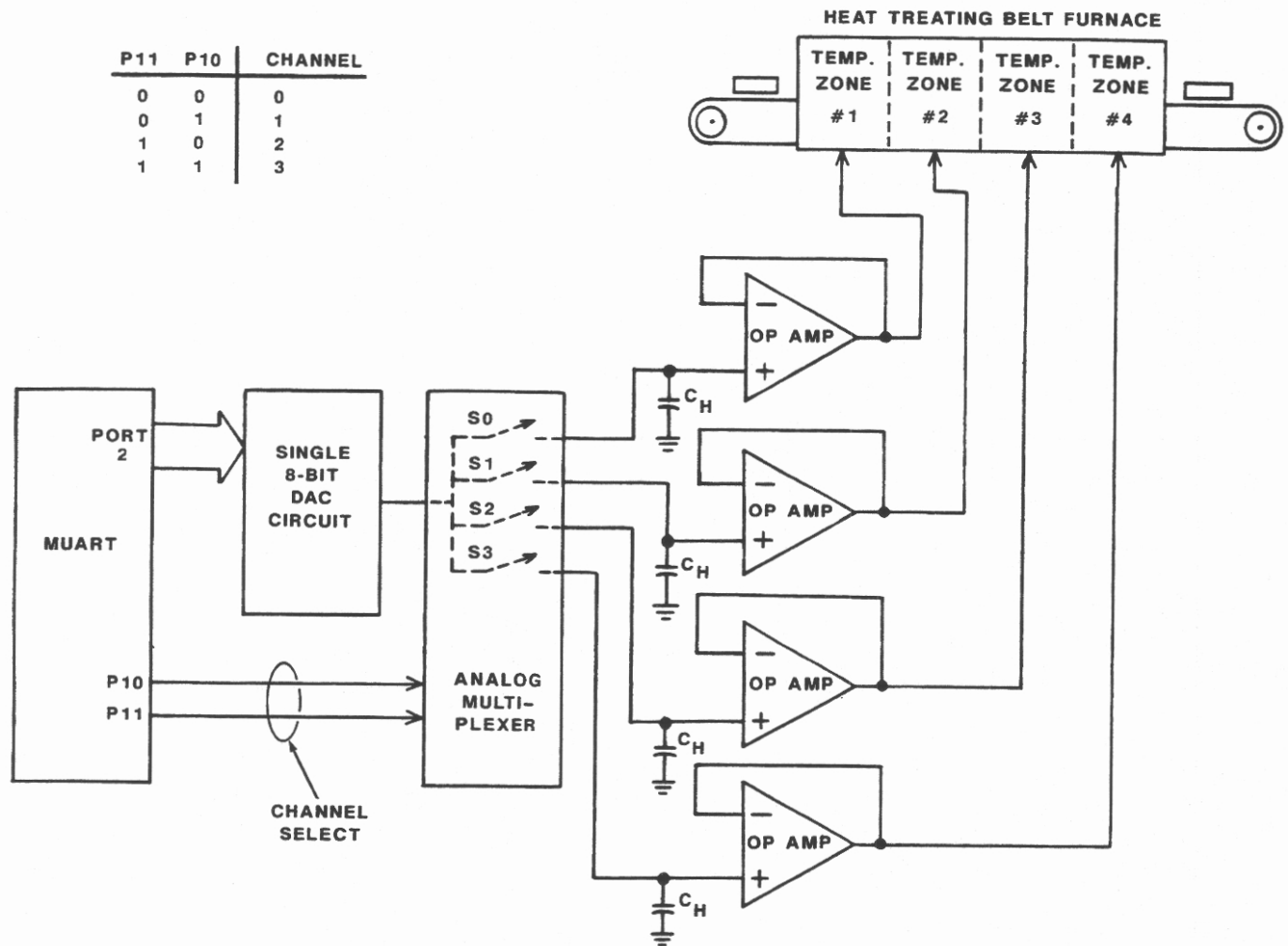


Figure 6-21

The 8-bit DAC output is multiplexed using an analog multiplexer and held using a voltage follower op amp circuit.

## ANALOG MULTIPLEXER

The analog multiplexer directs, or channels, the DAC output signal to one of its output lines. The particular output line selected is determined by the logic present on the channel-select lines of the multiplexer. In Figure 6-21, we have used the first four Port 1 output lines, *P10-P11*, to supply the channel select logic to the analog multiplexer.

This multiplexer is a typical four-channel analog multiplexer. From the truth table in Figure 6-21, you can see how the digital channel select logic from Port 1 is used to select one of four analog channels. Consequently, one switch can be closed and the other three open by the logic written to Port 1. The DAC output is then passed through the selected analog switch to a sample/hold device.

To control a given temperature zone, the microprocessor must be programmed to write the temperature control value to the DAC via Port 2 of the MUART, then write the channel-select logic to Port 1 of the MUART to select the desired control zone.

## SAMPLE/HOLD

The sample/hold circuits shown in Figure 6-21 can be nothing more than an op amp and a hold capacitor, labeled  $C_H$ . The op amp must be connected as a voltage, or source follower. In other words, the op amp acts as a unity gain non-inverting amplifier. Consequently, the op amp output follows its input. Notice that each input signal is applied to the non-inverting input line of the op amp. The output of the op amp is fed back directly to the inverting input line of the op amp. Connected in this way, the output of the op amp is equal to its input voltage. However, the input impedance of the voltage follower is very high and its output impedance very low. Consequently, it acts as an ideal impedance buffer between its input and output.

An output from the analog multiplexer in Figure 6-21 charges the respective hold capacitor  $C_H$  to the DAC output voltage level. The hold capacitor will remain charged for several minute or hours (depending on its value), since it "sees" the very high input impedance of the voltage follower circuit. In addition, since the temperature control values will most likely be stored in RAM, the microprocessor can be programmed to periodically refresh the capacitor charges if a small leakage is experienced over extended time periods. Thus, a control voltage provided by the DAC is stored by the sample/hold circuit for a control zone until it is changed by the microprocessor.



The value and type of hold capacitor determines the length of time that the output level is held. Capacitor values from 100 picofarads to 100 microfarad are typical. It is also recommended that polycarbonate, polystyrene, polypropylene, or Teflon<sup>®</sup> capacitors be used because of their very high leakage resistance properties.

You can eliminate the analog multiplexer in Figure 6-21 by using special sample/hold devices instead of op amps as shown in Figure 6-22. Notice from Figure 6-22 that each device has its own separate logic input control line. A logic one on this line allows the respective hold capacitor,  $C_H$ , to be charged to the analog input level. By connecting the sample/hold device control lines to port B as shown in Figure 6-22, you can select which sample/hold device will receive the DAC output by the logic written to Port 1 lines P10-P13.

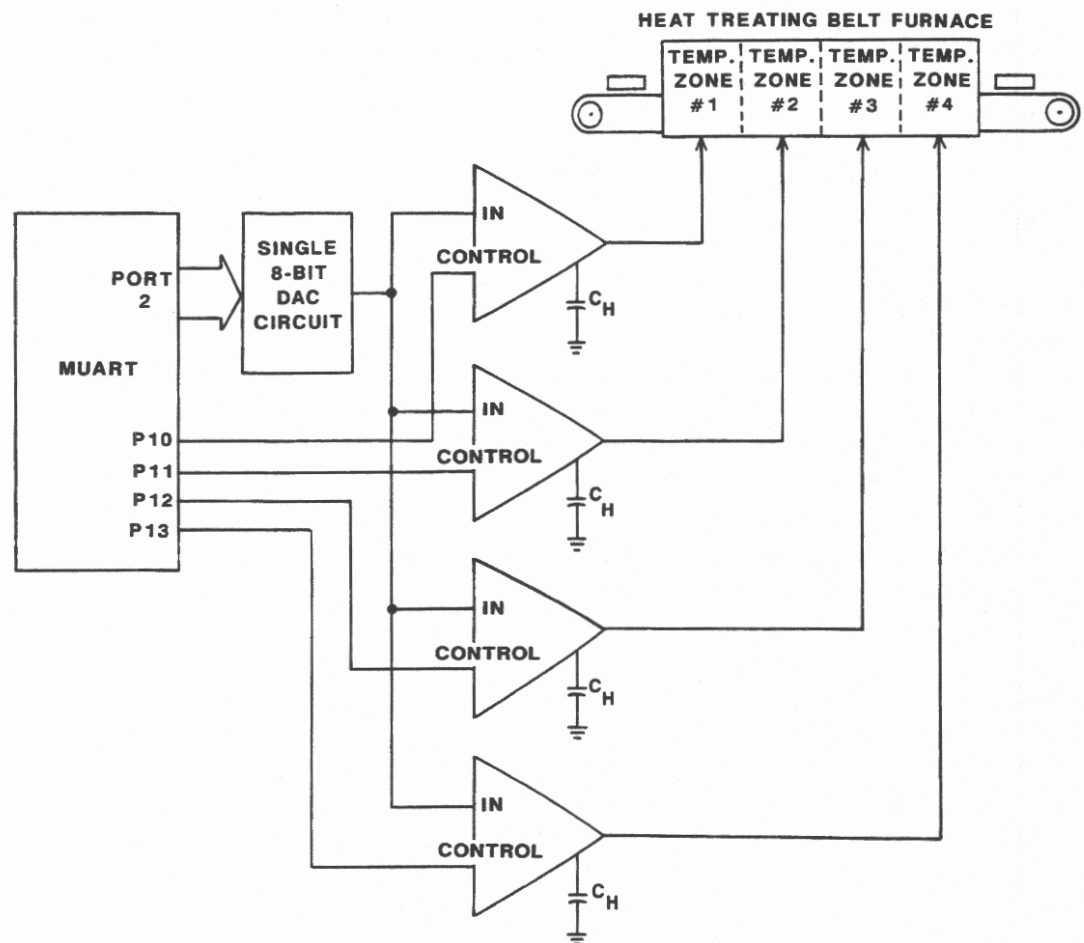


Figure 6-22

The DAC output can be multiplexed, sampled, and held by special sample/hold devices.

Sample/hold devices, like the LF398, are actually special purpose op amp circuits. They are available in dual in-line packages and operate on supply voltages from  $\pm 5$  V to  $\pm 15$  V. The hold capacitor, CH, is not supplied with the device and must be connected externally. Typically, the hold capacitor value is .01 microfarads and should be a polycarbonate, polystyrene, polypropylene, or Teflon<sup>®</sup> capacitor.

## Self-Test Review

6. Explain the major programming steps required to generate a square waveform from a microprocessor-controlled DAC circuit.
7. How would you change the duty cycle of a square waveform output from a microprocessor-controlled DAC circuit?
8. How can you control the frequency of an output waveform generated by a microprocessor-controlled DAC circuit?
9. How can you adjust the amplitude of an output waveform generated by a microprocessor-controlled DAC circuit?
10. What type of DAC is required to construct a programmable gain amplifier and attenuator?
11. Where is the input signal applied to a programmable gain amplifier and attenuator?
12. What adjusts the gain and/or attenuation level of a programmable gain amplifier and attenuator?
13. How can you control the direction and speed of a DC motor?
14. What is a servo amplifier and why is it required in a DC motor control circuit?
15. Describe the major parts of a variable speed DC motor control circuit that uses a microprocessor-controlled DAC circuit.
16. What is a comparator and how is it used in a DC motor control circuit?
17. What two general methods can be used to provide microprocessor/DAC control of a process?
18. What is an analog multiplexer?
19. What is a sample/hold circuit?

## Answers

6. The following major programming steps are required to generate a square waveform from a microprocessor-controlled DAC circuit:
  - Store a maximum value to the DAC to obtain the maximum desired output level.
  - Delay.
  - Store a minimum value to the DAC to obtain the minimum desired output level.
  - Delay.
  - Repeat.
7. The duty cycle of a square waveform output can be changed by changing the delay periods in the waveform generation routine. Equal delay periods will provide a 50% duty cycle. Unequal delay periods will provide duty cycles other than 50%.
8. You can control the frequency of an output waveform with the delay periods in the waveform generation routine. In general, longer delays create a lower output frequency.
9. You can adjust the amplitude of an output waveform with software or hardware. The amplitude can be adjusted with software by changing the maximum and minimum values stored to the DAC. With hardware, the amplitude can be adjusted by changing the values of RREF or Rf in the DAC circuit.
10. A multiplying DAC is required to construct a programmable gain amplifier and attenuator.
11. The input signal is applied to the reference voltage input of the DAC in a programmable gain amplifier and attenuator.
12. The gain and/or attenuation level of a programmable gain amplifier and attenuator is adjusted by the digital input value to the DAC.
13. The direction of rotation of a DC motor is controlled by the polarity of the control voltage. The speed of a DC motor is controlled by the amount of control voltage applied to the motor.
14. A servo amplifier is a power amplifier that is required in a motor control circuit to supply the current levels required by the motor.

15. A variable speed DC motor control circuit must consist of a unipolar DAC circuit, difference amplifier, and servo amplifier. In addition, a linear feedback potentiometer must be geared to the motor shaft to provide motor position information. (See Figure 6-16).
16. A comparator is an op amp circuit that compares two input voltage values. The output of a comparator will be a constant positive, negative, or zero level, depending on the greater than, less than, or equal status of the two input voltage values. A comparator is used in a DC motor control circuit to compare the DAC output control voltage to the output of a motor position transducer. The comparator output drives a servo amplifier which, in turn, drives the motor. (See Figure 6-14.)
17. When a microprocessor/DAC circuit is used to control a process, you can use multiple DACs - one for each control function. Another, more economical, method is to use a single DAC along with an analog multiplexer and sample/hold circuits.
18. An analog multiplexer is nothing more than a series of analog switches. A single input is directed to one of several output lines by applying the appropriate logic to the channel select lines of the multiplexer. The MC14066B is a typical analog multiplexing device.
19. A sample/hold device is simply a voltage memory, or storage, device. A sample/hold circuit can be constructed using a voltage follower op amp circuit and a hold capacitor. In addition, special sample/hold devices, like the LF398, can be purchased from several semiconductor manufacturers.

## INTERFACING TO A/D CONVERTERS

Analog signals can be converted to parallel digital data using devices called ADCs, or serial digital data using V/F converter devices. You are now ready to learn how to interface these devices to a microprocessor system. We will use a MUART as the interface between the MPU and converter. The flexibility and control features of the MUART provide a definite advantage over combinational logic when interfacing to analog-to-digital converter devices.

In this section, you will expand your knowledge of the MUART in order to provide the handshake control of ADC devices. In addition, you will learn how to write the MPU software required to count the serial output pulses generated by V/F converter devices. Finally, you will see how analog multiplexers and sample/hold devices are used to enhance the data acquisition capabilities of all analog-to-digital converters.

### Interfacing and Controlling ADC Devices

As we mentioned in the introduction to this section, we will use a MUART as the interface between the microprocessor and ADC device. The circuit in Figure 6-23 shows one way that an 8-bit ADC can be interfaced to a microprocessor via a MUART. Here, we have chosen to use Port 2 of the MUART. The eight digital output lines of the ADC are connected to the eight Port 2 data lines. Consequently, port 2 must be configured as an input port when the MUART is initialized.

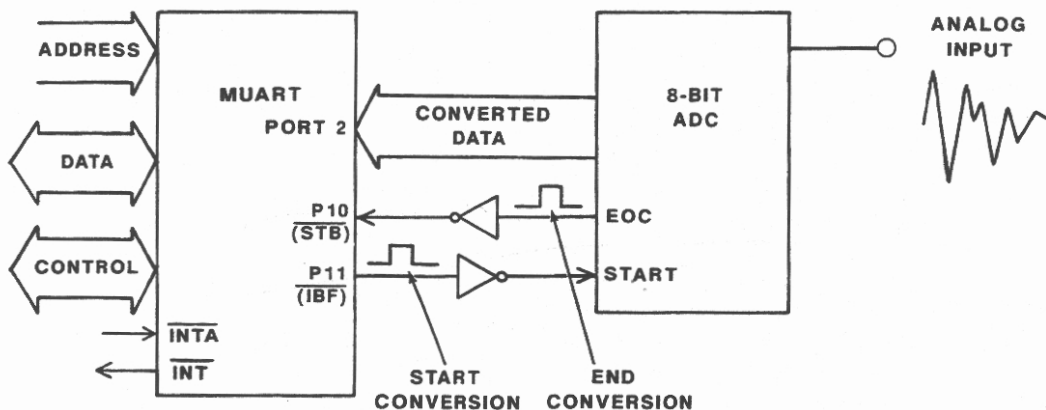


Figure 6-23

The MUART can control the conversion process with its control lines.

## HANDSHAKE CONTROL OF AN ADC

Now, let's see how this type of control can be provided with the MUART interface circuit in Figure 6-23.

The handshaking control of the ADC in Figure 6-23 can be provided by configuring the MUART Mode Register for the input handshake mode. This configuration requires that the value 04H be written to the Mode Register. Recall that input handshaking employs Port 2 as an input port and the *P10* and *P11* lines of Port 1 as handshake control lines. Thus, the ADC output data lines are connected to Port 2 in Figure 6-23. Then, the *P10* and *P11* Port 1 lines are used to provide the necessary handshaking to control the conversion process. As you can see, the *P11* ( $\overline{IBF}$ ) control line is used to start a conversion via the ADC *START* line, while the *P10* ( $\overline{STB}$ ) control line receives the end-of-conversion signal via the ADC *EOC* line. Inverters are used on the control lines to invert the control signals to their correct logic. The ADC *START* and *EOC* lines are both active high, while the MUART  $\overline{IBF}$  and  $\overline{STB}$  lines are active low.

Now, here's how the handshake works. After the MUART Mode Register has been initialized with the value 04H, the Level 7 interrupt must be enabled by writing the value 80H to the MUART Set Interrupts Register. Recall that a level 7 interrupt is used for the handshake operation within the MUART. When a conversion is to be started, the MPU must read Port 2. This starts the handshaking process. The read Port 2 operation generates a *START* pulse via the MUART *P11* ( $\overline{IBF}$ ) line.

Once the ADC conversion process is started, the MPU is free to go off and perform other tasks until the ADC completes the conversion. When the conversion is complete, the ADC will activate its *EOC* line, which is connected to the *P10* ( $\overline{STB}$ ) input line of the MUART. When the *P10* ( $\overline{STB}$ ) line is activated by the ADC *EOC* line, a Level 7 interrupt is generated to the MPU via the MUART *INT* line. The MPU must then read Port 2 to obtain the converted value. The read Port 2 operation generates another *START* pulse via the MUART *P11* ( $\overline{IBF}$ ) line and this handshaking process repeats to obtain another converted value, and so on until the MPU stops reading Port 2.

Flowcharts of the necessary control software are provided in Figure 6-24. Here you see the main program configures the MUART for the input handshaking operation by writing the value 04H to the Mode Register and the value 80H to the Set Interrupts Register. Port 2 is then read to start the conversion process. In addition, the 8085 interrupts are enabled. Since the MUART handles the handshaking via a Level 7 interrupt, the RST7 interrupt must be employed within the 8085. When the conversion is complete, the RST7 interrupt service routine reads Port 2 to obtain the converted value and stores it in memory. The read Port 2 operation also starts another conversion.

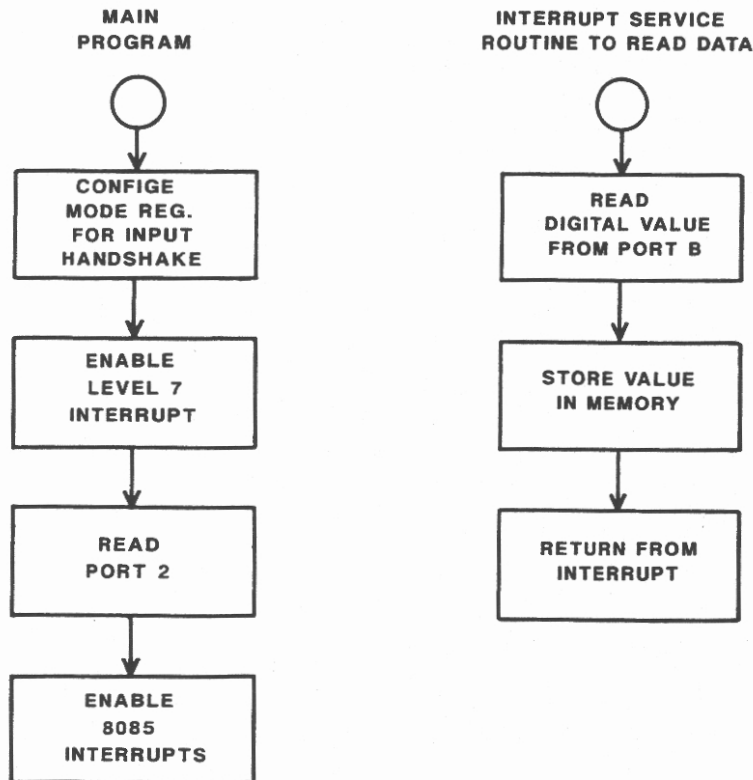


Figure 6-24

A flowchart summary of the software required for handshake control of an ADC via the MUART.

As you are aware, the control process just described is called a **handshake**. The term handshake is used because two-way communication takes place between the MUART and ADC over the two MUART control lines. Up to this point, you have not had a need to provide handshaking between the MUART and an external device. You simply read a MUART input port to obtain data from an input device and write data to a MUART output port to output data to an output device. So, you're probably wondering why you need to provide handshaking for an ADC interface. Why not simply read the data continuously as it becomes available to the MUART? Recall that no handshaking was required to control a DAC. You simply write data to the DAC via a MUART output port. The DAC then converts the digital data as fast as it can be provided by the MPU.

The reason that handshaking must be used in the ADC interface is that ADCs are relatively slow devices. On the other hand, a DAC is a relatively fast device. The key term here is relative speed. If an external device, like a DAC, can operate faster than the MPU, then no handshaking is required. However, if an external device is slower than the MPU, some sort of handshake control is required.



As we said earlier, because of its speed, a DAC can make conversions as fast as the MPU can supply data to it. However, most ADCs operate in the millisecond range while the MPU operates in the microsecond range. Therefore, the MPU must wait to receive data between conversions. With the handshaking procedure, the MPU starts the conversion by activating the ADC *START* line. The MPU is then free to perform other tasks while the ADC is making the conversion. When the conversion is complete, the ADC activates its *EOC* line and interrupts the MPU via the MUART to read the converted data.

Most common *microprocessor compatible* ADCs have internal output latches. However, if the ADC output is not latched, you must provide a latch between the ADC and MUART as shown in Figure 6-25. Recall that for input operations, the MUART does not latch data. The latch is required since, without it, the unlatched ADC output data would be lost before the MPU could service the end of conversion interrupt and read the data. Notice from Figure 6-25 that the *EOC* signal enables the external latch and activates the *CBI* input line simultaneously.

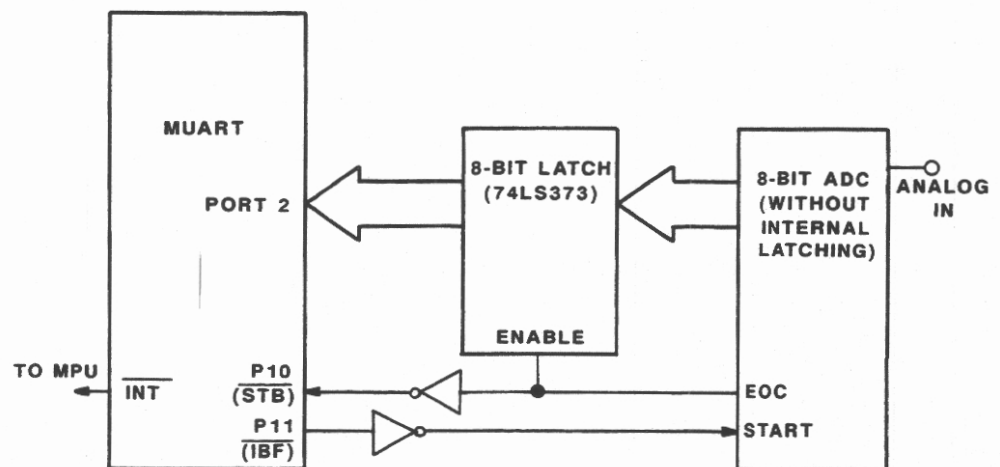


Figure 6-25

External latching must be used for the interface if the ADC is not microprocessor compatible.

Up to this point, I have only discussed an 8-bit ADC interface. Suppose the ADC output is less or more than eight bits. The less-than-8-bit case is easy. You simply use fewer MUART port lines. Hence, a 6-bit ADC would only require that six MUART port data lines be used. The control line connections and operation would be the same as for an 8-bit ADC.

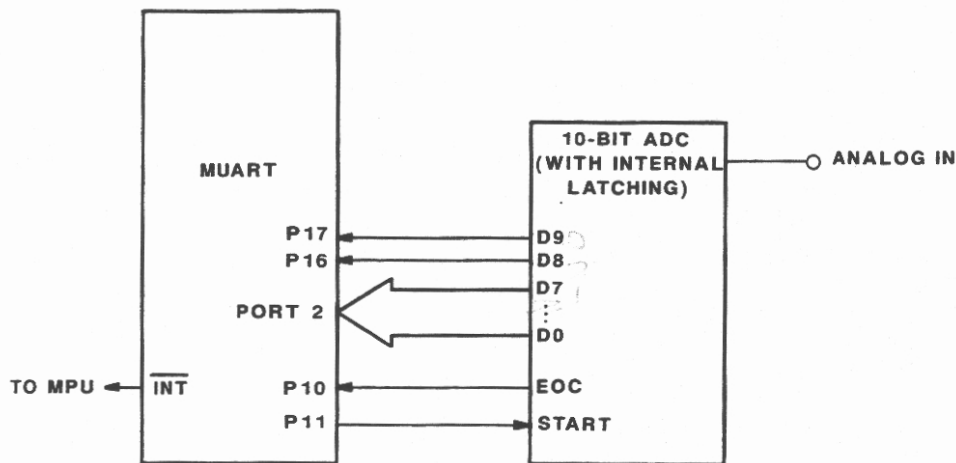


Figure 6-26

A 10-bit ADC with internal latching connected to the MUART.

Now, suppose you want to achieve higher digital resolution, and higher accuracy of the analog signal. Then, a 10-bit, 12-bit or even 14-bit ADC must be used. This presents an interfacing problem because the MUART ports are only 8-bit ports. The solution to the problem is to use both MUART ports as illustrated by the 10-bit ADC interface circuit shown in Figure 6-26. Here, Port 2 is used to input the eight least significant bits,  $D0-D7$ . The two most significant bits are input by two Port 1 lines ( $P17$  and  $P16$ ).

Now, the problem is to get the 10-bit converted value from the MUART into the MPU. The 10-bit converted value can then be input using the following algorithm:

BEGIN

Read Port 1 into A.  
 Rotate A right six times.  
 Mask-off six most significant bits of A.  
 Move A to B.  
 Read Port 2 into A.  
 Move A to C.  
 Move BC pair to memory.

END.

Notice that the Port 1 value is first read into the accumulator. Now, since the *P17* and *P16* port lines are used to input the two most significant bits of the 10-bit value, the two most significant bits of the accumulator will contain the two most significant bits of the 10-bit value. These bits need to be shifted down to the two least significant bit locations of the accumulator. This is accomplished by rotating the accumulator right six times. Next, the six most significant bits of the accumulator are masked-off to produce all 0's in these bit locations. The accumulator contents are then moved to the B register. Thus, the B register contains the two most significant bits of the 10-bit value in its two least significant bit locations.

Port 2 is then read into the accumulator and transferred to register C. Now, the BC register pair contains the 10-bit converted value. This value can then be stored in memory at two consecutive memory addresses. Do you feel that you could code the above algorithm? This will be left as an exercise at the end of this section.

Finally, a simpler interface, without handshake control, can be provided by jumpering the ADC *EOC* line to its *START* line as shown in Figure 6-27. With this interface, the MPU has no control over the conversion process and the ADC will provide continuous conversions. The MPU can then read the data at regular intervals. However, remember that the ADC cannot convert as fast as the MPU can read the data. Consequently, a delay must be provided between MPU read operations. Many times, this approach is less efficient than the handshake approach since the MPU must provide the time delay via software. Software time delays waste MPU time!

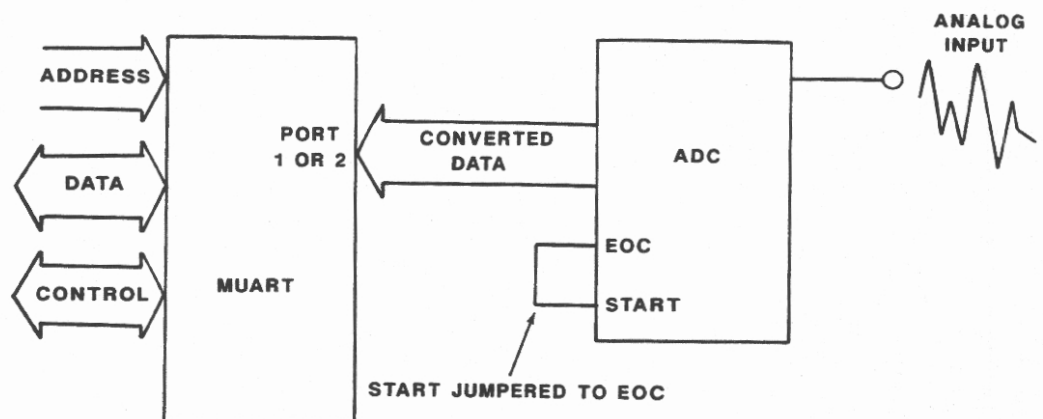


Figure 6-27

The ADC provides continuous conversions when its *START* line is jumpered to its *EOC* line. However, the MPU has no control over the conversion process.

## Interfacing and Controlling V/F Converters

A V/F converter converts an analog voltage level to a series of digital pulses whose frequency is proportional to the analog input level. To interface a V/F converter to a microprocessor system, the output pulses of the converter are counted over a specific time period. The pulse count is then converted to a parallel digital quantity that can be used by the microprocessor. In other words, you must provide a serial-to-parallel conversion of the V/F converter output.

As with most microprocessor design problems, the solution can be accomplished with internal software or external hardware. You will find that there is always a trade-off between software and hardware in microprocessor-based designs. Less external hardware requires more internal software, and vice-versa. Therefore, the engineer or technician must decide on the design approach that results in the most efficient and economical system that satisfies the application requirements. Let's take a close look at both the hardware and software approaches for interfacing V/F converters. You will find that the trade-offs encountered here are typical of many microprocessor interfacing problems.

### V/F CONVERTER INTERFACING USING EXTERNAL HARDWARE TECHNIQUES

A hardware interface for a V/F converter is shown in Figure 6-28. The key ingredients here are a timer, MUART, and an 8-bit binary counter. Here's how it works: The TTL output of the V/F converter is fed into one side of a NAND gate. It is assumed that the V/F converters provide active low output pulses. If this is the case, an inverter must be placed between the V/F converter output and the NAND gate input.

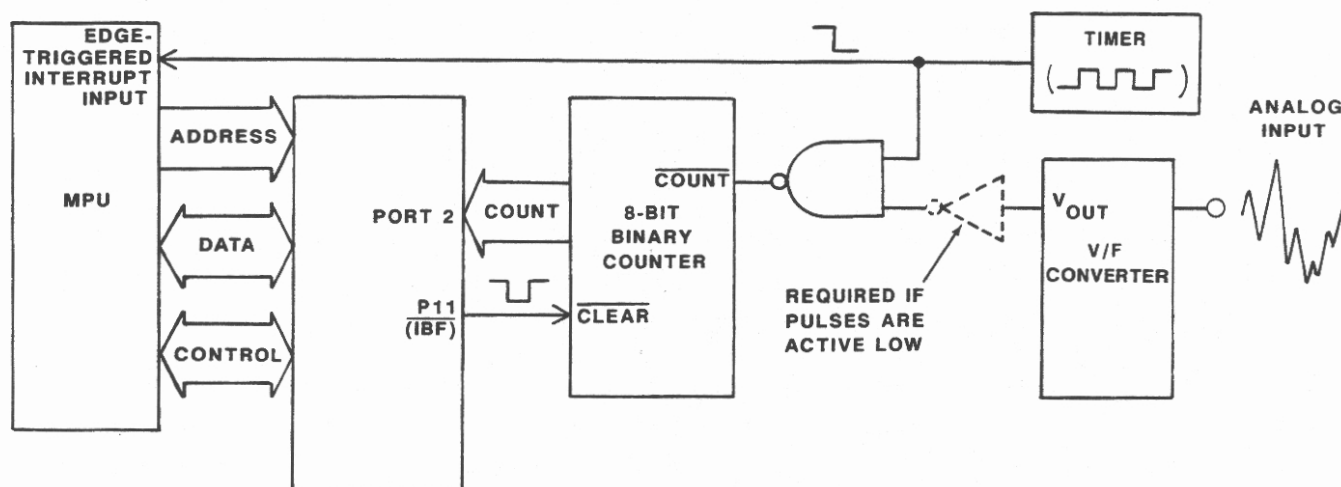


Figure 6-28

A hardware interface for a V/F converter.

The TTL output of the external timer is applied to the other side of the NAND gate. Now, refer to the associated timing diagram in Figure 6-29. The frequency of the timer output determines the count period. When the timer output goes high, the output of the NAND gate goes low for each active high pulse produced by the V/F converter. Notice from Figure 6-29 that the binary counter counts on a low pulse from the NAND gate. Consequently, during the count period (when the timer output is high) the counter will increment, by one, for each pulse provided by the V/F converter.

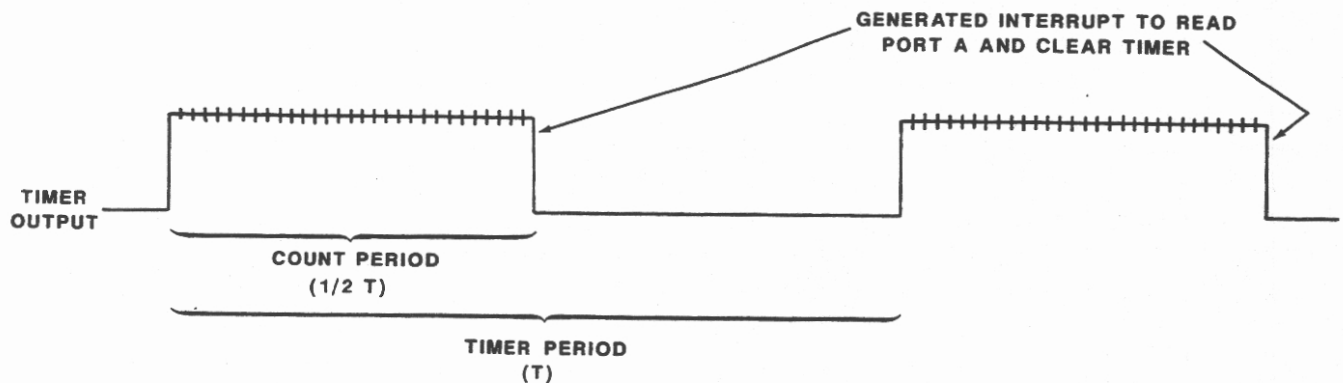


Figure 6-29

A timing diagram of the V/F converter interface.

The binary counter continues to count for each output pulse of the V/F converter as long as the timer output remains high. When the timer output goes low, two things happen: (1) the counter stops counting, and (2) an interrupt is generated to the MPU. The counter stops counting since, when the timer output is low, the output of the NAND gate remains high, regardless of the pulses generated by the V/F converter. In addition, an interrupt is generated to the MPU by the high-to-low transition of the timer output.

When the interrupt is generated, the MPU reads Port 2 of the MUART to obtain the binary count. The count value is proportional to the output frequency of the V/F converter which, in turn, is proportional to the analog input of the V/F converter. Furthermore, notice from Figure 6-28 that the *P11 (IBF)* output control line of the MUART is connected to the *CLEAR* input line of the binary counter. The MUART Mode Register can be configured such that an active low pulse is generated on the *P11 (IBF)* output control line to clear the counter when a read-Port-2 operation takes place. To do this, the value 04H is stored in the Port 2 control register when the MUART is initialized. With this configuration, the counter will be cleared each time a new count is obtained by reading Port 2. Thus, the counter is reset to zero and ready to begin a new count when the timer output returns high.

It should be obvious that the count period in Figure 6-29 is dependent upon the output frequency of the timer. The conversion resolution is, in turn, dependent upon the count period. For example, suppose you are using a 10 kHz V/F converter and the timer output frequency is 60 Hz. With a 60 Hz timer frequency, the count period is  $\frac{1}{60 \text{ Hz}} + 2$ , or 8.33 milliseconds. Now, the 10 kHz V/F converter will generate a maximum of one pulse every  $\frac{1}{10 \text{ kHz}}$ , or .1 milliseconds for a full-scale analog input voltage value. Consequently, a full-scale analog input voltage level to the V/F converter provides a maximum count of  $\frac{8.33 \text{ milliseconds}}{.1 \text{ milliseconds}}$ , or approximately  $83_{10}$ . Thus, as the analog input level ranges from 0 to full scale, the pulse count will range proportionally from 0 to  $83_{10}$ .

Now, suppose the timer output frequency is 20 Hz. Then, the count period would be  $\frac{1}{20 \text{ Hz}} + 2$ , or 25 milliseconds. With this count period, a full-scale analog input to the V/F converter produces a count of  $\frac{25 \text{ milliseconds}}{.1 \text{ milliseconds}}$ , or  $250_{10}$ . The count now ranges proportionally from 0 to  $250_{10}$  as the analog input level ranges from 0 to full scale. Notice that the conversion resolution has been tripled by increasing the count period. A word of caution - the maximum count must not be over the value  $256_{10}$ , since only an 8-bit counter is used in this circuit.

If increased resolution is desired, a larger counter must be used. However, this presents another problem in that the MPU must perform multi-byte operations via the MUART. You will find that an 8-bit count provides enough resolution for most sensing and control applications. For example, if a 10 kHz V/F converter has an analog input range of 0 to 10 volts, an 8-bit counter and a timer frequency of 20 Hz could resolve to 1-part-in-250, or  $\frac{10\text{V}}{250} = .04$  volts. If this is not enough resolution, consider using a 100 kHz V/F converter rather than increasing the size of the counter.

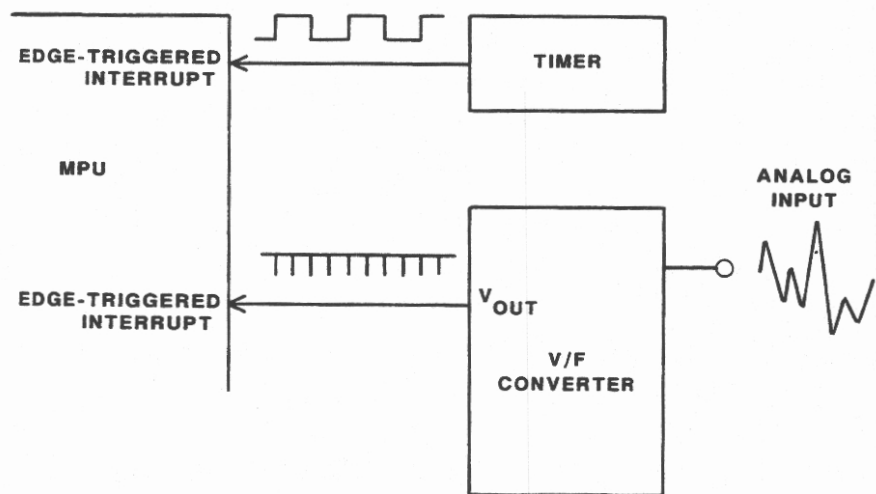


Figure 6-30

A simple V/F converter interface that uses software to count the converter pulses.

## V/F CONVERTER INTERFACING USING SOFTWARE TECHNIQUES

You can eliminate most of the V/F converter interface hardware by using more MPU software. Notice from the circuit in Figure 6-30 that both the MUART and binary counter have been eliminated from the interface. Instead, the V/F converter output is connected directly to an edge-triggered interrupt input of the MPU. Like before, the output of the timer is connected to another edge-triggered interrupt input of the MPU. Now, refer to the timing diagram in Figure 6-31.

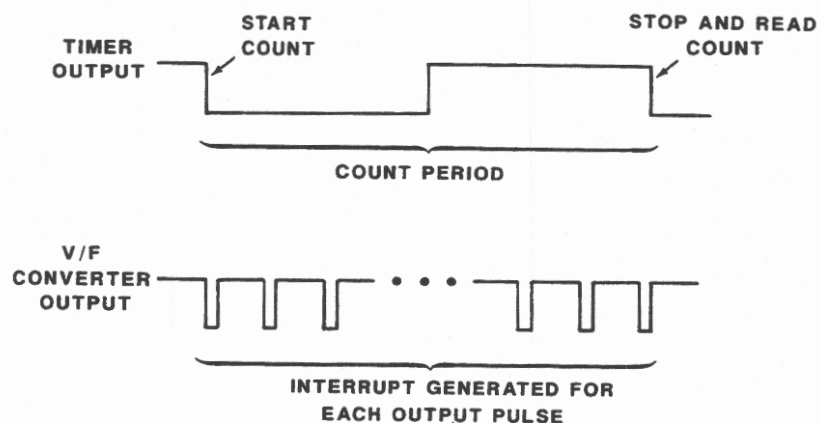


Figure 6-31

A timing diagram for the software controlled V/F converter interface.

This time, we have assumed that the V/F converter output pulses are active low. Thus, an interrupt request is generated to the MPU each time a pulse is generated by the V/F converter. The MPU interrupt request service routine increments a memory location each time a pulse is received. In other words, the MPU is counting the V/F converter output pulses by means of an interrupt service routine which increments a memory location. The pulses are counted for the duration of the count period which is determined by the external timer frequency.

Each time the timer activates the interrupt input with a high-to-low transition, the MPU must obtain the accumulated V/F converter pulse count by reading the designated memory location. After the count is read, the MPU must clear the pulse count memory location to prepare for a new pulse count.

Notice that the timer interrupt is activated once every cycle of the external timer output. Consequently, the pulse count is allowed to accumulate for the duration of the timer output cycle. If the frequency of the timer output is 60 Hz, the count

would accumulate for  $\frac{1}{60 \text{ Hz}}$ , or 16.66 milliseconds. With a 10 kHz V/F converter, the full-scale count would be  $\frac{16.66 \text{ milliseconds}}{.1 \text{ milliseconds}}$ , or  $166_{10}$ . Thus, as the

analog input ranges from 0 to full scale, the pulse count will range proportionally between 0 and  $166_{10}$ . As before, a lower clock frequency will provide a longer count period which, in turn, provides increased resolution. However, to avoid extra software, the maximum pulse count must be less than  $256_{10}$  since an 8-bit memory location is being used as the pulse counter.

As you can conclude from the above discussion, the software approach requires two interrupt service routines. These routines are illustrated by the flowcharts in Figure 6-32.

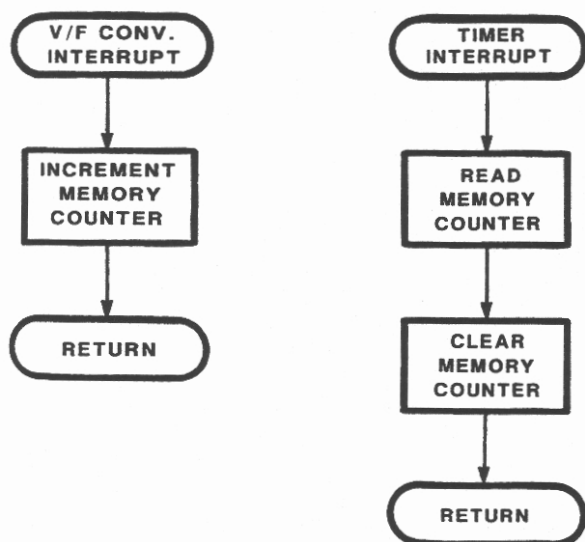


Figure 6-32

Interrupt service routines required for software V/F converter interfacing technique.



We should mention one other important point when interfacing to V/F converters. V/F converters are useful for remote sensing applications. In such an application, electrical isolation can be easily provided between the V/F converter and the digital circuit using an optical transmission path as illustrated in Figure 6-33. Many times, it is desirable to isolate the analog sensing circuit from the digital circuit. An opto-isolator or fiber optic system can be used to achieve several kilovolts of electrical isolation. In addition, the two circuit grounds are electrically isolated, thereby preventing possible lethal situations.

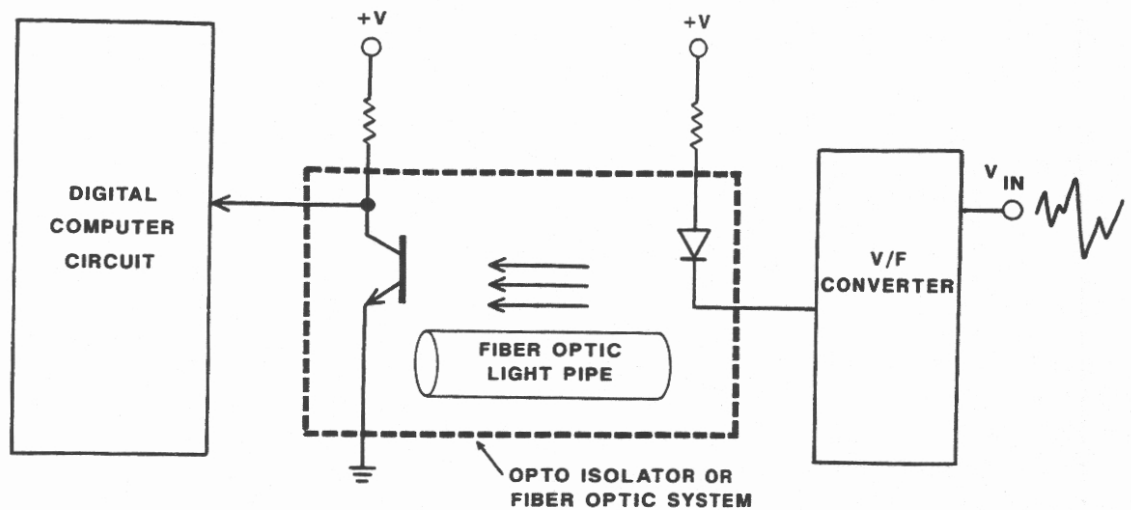


Figure 6-33

An opto-isolator or fiber optic system can be used to electrically isolate the analog sensing circuit from the digital computer circuit.

## Self-Test Review

20. Describe how an 8-bit ADC is connected to the MUART.
21. Why is handshaking usually required between the MPU and ADC?
22. What registers must be initialized within the MUART to perform an ADC handshake?
23. How does the MPU start the handshaking process when the MUART is configured for an input handshake?
24. How does the MPU detect when the ADC has completed a conversion using the MUART in its input handshaking mode?
25. Write the code for the 10-bit ADC algorithm given in this section. Assume the MUART is decoded for port addresses 50H through 5FH.
26. How can an ADC be made to provide continuous conversions?
27. What are the key ingredients for a V/F converter hardware interface?
28. What is the function of a timer in a V/F converter interface?
29. How must you connect a V/F converter to the MPU when using software techniques to count the converter output pulses?
30. Suppose you are using a software technique to count the output pulses of a 10 kHz V/F converter. The output frequency of the pulse count timer is 40 Hz. What count value corresponds to a full-scale input to the converter?

## Answers

20. The eight output lines of an 8-bit ADC must be connected to Port 2 of the MUART. The *START* line of the ADC is connected to the *P11* ( $\overline{IBF}$ ) line of the MUART and the *EOC* line of the ADC is connected to the *P10* ( $\overline{STB}$ ) line of the MUART. (See Figure 6-23).
21. Handshaking is necessary because ADCs are relatively slow devices. Handshaking allows the MPU to control the data conversion process with efficiency.
22. The Mode Register must be initialized for input handshaking and the Set Interrupts register must be initialized to enable the Level 7 interrupt.
23. By reading Port 2.
24. By the receipt on an *RST7* interrupt from the MUART.
25. IN 59H  
RAR  
RAR  
RAR  
RAR  
RAR  
RAR  
ANI 03H  
MOV B,A  
IN 59H  
MOV C,A  
MOV M,B  
MOV M,C
26. An ADC can be made to provide continuous conversions by jumpering its *EOC* output line to its *START* input line.
27. The key ingredients for a V/F converter hardware interface are a timer, MUART, and a binary counter.

28. The function of a timer in a V/F converter interface is to control the pulse count period, or measurement time, which in turn determines the conversion resolution.
29. The TTL output of the V/F converter is connected directly to one of the MPU interrupt input lines. The output of a timer is connected to another one of the MPU's interrupt input lines.
30. Since the frequency of the timer output is 40 Hz, the count would accumulate for  $\frac{1}{40 \text{ Hz}}$ , or 25 milliseconds. With a 10 kHz V/F converter, the full-scale count value must be  $\frac{25 \text{ milliseconds}}{.1 \text{ milliseconds}}$ , or  $250_{10}$ .

## MICROPROCESSOR APPLICATIONS USING A/D CONVERTERS

The three major areas where analog-to-digital converters are used are in digital instrumentation, data acquisition, and industrial control systems. In this section, you will explore each of these major application areas. You will find that the analog-to-digital converter is at the heart of most digital instruments. Such instruments are rapidly replacing analog instrumentation throughout industry. You will also see how analog-to-digital converters and their associated components are applied to data acquisition and industrial control systems. Analog-to-digital converters are required in these systems so that digital computers can sense and control "real world" events.

### Instrumentation

Digital instruments are rapidly replacing the meter movement analog type instruments. In many cases, digital instruments are more accurate, easier to read, faster, and less expensive than their analog counterparts. In addition, many digital instruments can be easily interfaced to microprocessor-based control systems for data acquisition, analysis, and control purposes. You will usually find an analog-to-digital converter at the heart of a digital instrument as shown in Figure 6-34.

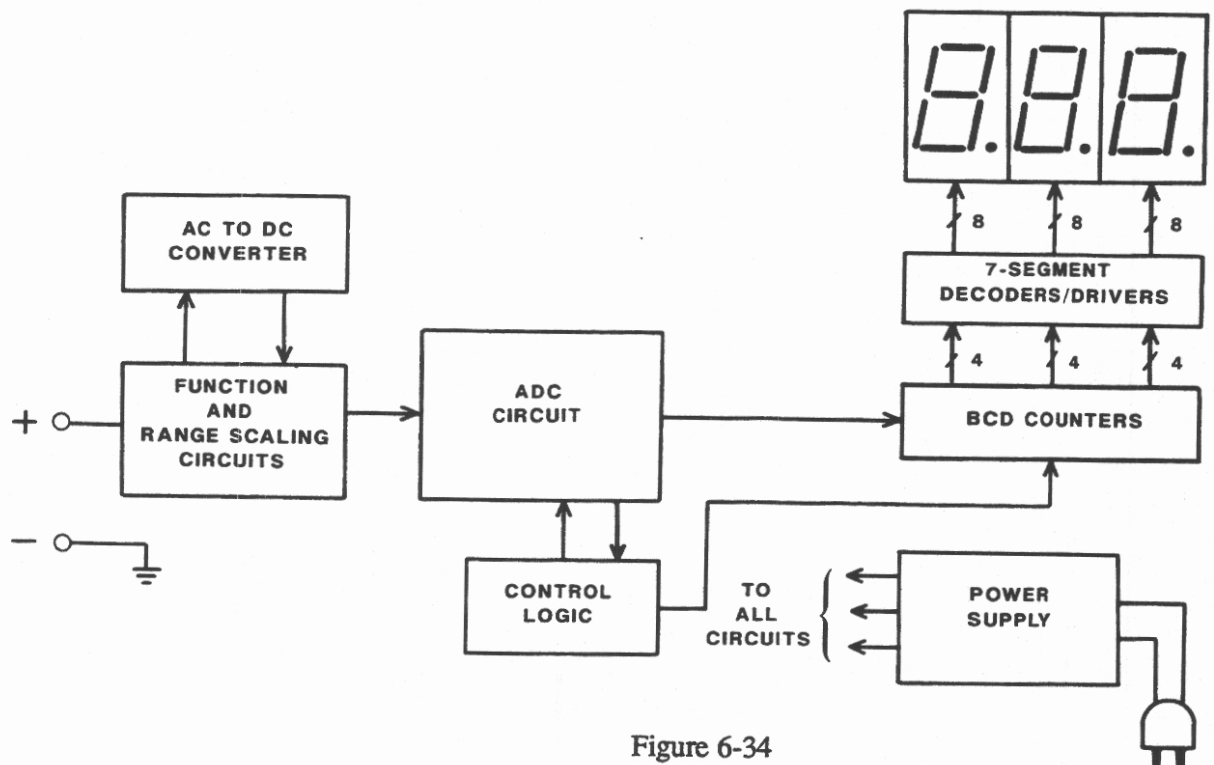


Figure 6-34

A typical inexpensive digital voltmeter, or DVM, circuit.

## Data Acquisition

We will begin this material by defining what we mean by a microprocessor data acquisition system.

**microprocessor data acquisition system:**

a microprocessor-based system consisting of analog-to-digital converters, analog multiplexers, sample/holds, signal conditioners and transducers/sensors that process one or more analog signals and converts them to digital form for use by the microprocessor.

A block diagram of a typical data acquisition system is shown in Figure 6-35. Up to this point, you have been acquainted with the analog-to-digital converter and interface sections of a data acquisition system. In addition, you observed the use of analog multiplexing and sample/hold circuits when we discussed process control applications of digital-to-analog converters. Now, you will expand your knowledge of analog multiplexers and sample/hold circuits as they apply to data acquisition systems. Then, you will observe several specific applications of such a system. You might say that, from a learning point of view, you have been building the data acquisition system shown in Figure 6-35 from the inside out. That is, from the MPU out to the transducer/sensor.

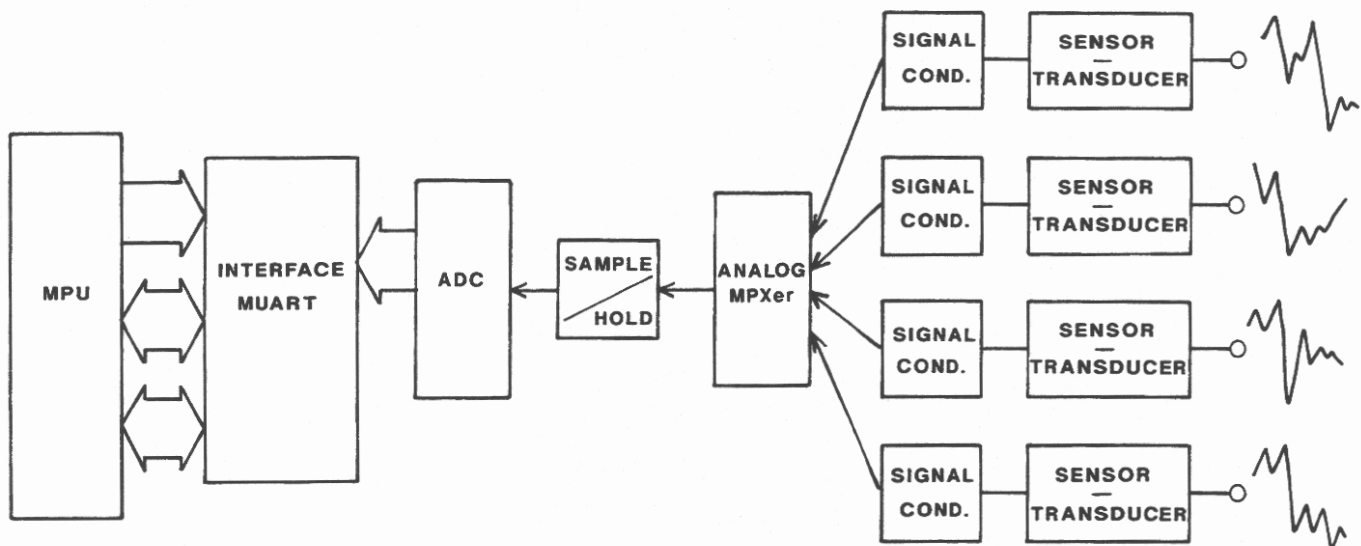


Figure 6-35

A typical microprocessor data acquisition system consists of analog-to-digital converters, analog multiplexers, sample/holds, signal conditioners, and transducer/sensors.

## ANALOG MULTIPLEXERS IN A DATA ACQUISITION SYSTEM

As illustrated in Figure 6-35, a typical data acquisition system collects data from several external transducer/sensors. The analog data supplied by each sensor/transducer must be converted to digital data to be analyzed by the microprocessor. You could provide a separate analog-to-digital converter for each analog signal to be converted. However, in many cases, the analog-to-digital converter is one of the most expensive devices in the system. It, therefore, becomes more economical to multiplex the analog signals into a single analog-to-digital converter.

An analog multiplexer allows two or more analog signals to share the same signal transmission path at different times. Analog multiplexers employ solid-state analog switches that, at the appropriate time, connect each input to a common output - often through a buffer amplifier as illustrated in Figure 6-36. The open or closed state of the analog switches is determined by a digital control signal.

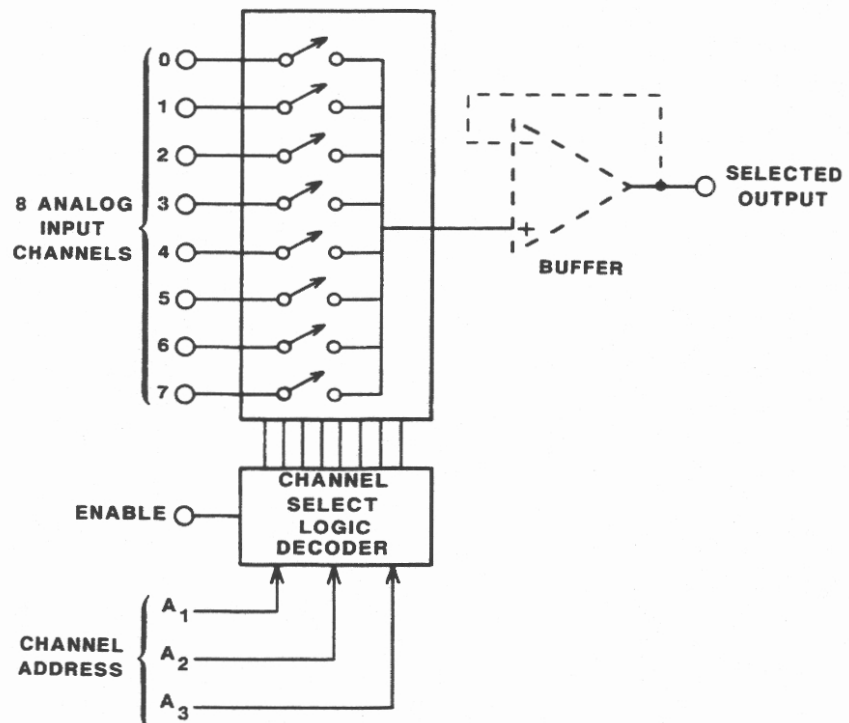


Figure 6-36

An analog multiplexer allows two or more analog signals to share a common transmission path.

In Figure 6-36, the digital control input has been labeled *CHANNEL ADDRESS*, since the applied digital logic results in the selection of the input channel. Channel selection is provided by an internal decoder circuit. This particular multiplexer is an 8-channel analog multiplexer. Thus one-of-eight input channels is selected by the channel select logic. In addition, the *ENABLE* line must be active for any channel to be passed to the output.

The single-ended output of an analog multiplexer can be connected directly to the input of an ADC as shown in Figure 6-37. We have assumed that the analog signal has been properly amplified and conditioned to be compatible with the ADC or V/F converter input prior to multiplexing. The 8-bit ADC in Figure 6-37 is interfaced to the MPU via a MUART as we discussed in the last section of this unit.

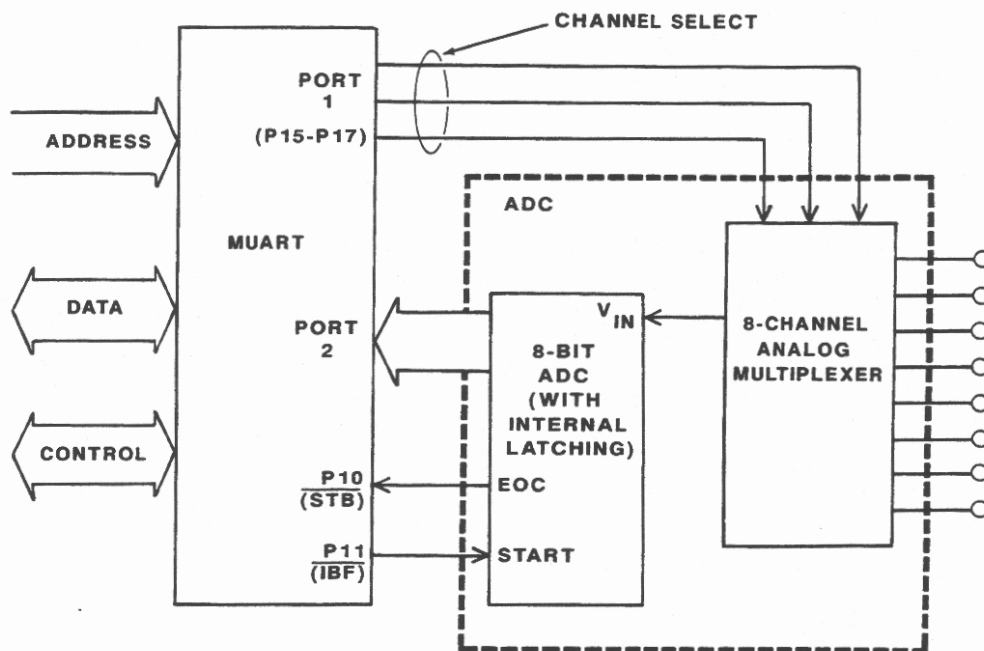


Figure 6-37

An 8-channel multiplexed input for an 8-bit ADC interface.

One of eight analog inputs is passed to the ADC through an 8-channel multiplexer. Channel selection is provided by using three of the MUART port lines as output lines. The MPU software can select any one of eight analog inputs by writing the proper logic to these MUART output lines. If more analog signals must be monitored, a larger multiplexer would be required. Of course, a larger multiplexer would require the use of more channel-select lines.



Many ADCs now provide the analog multiplexing capability internally, as indicated by the dashed outline in Figure 6-37. Many times, the cost of an ADC with an internal multiplexer is less than the total cost of an ADC and analog multiplexer purchased individually.

An analog multiplexer can also be used with a V/F converter. If the analog signals have been properly amplified and conditioned, the output of the multiplexer can be connected directly to the V/F converter input. The V/F converter would be interfaced to the MPU as discussed previously, and a MUART can be used to provide the channel-select logic to the multiplexer.

## SAMPLE/HOLD CIRCUITS IN A DATA ACQUISITION SYSTEM

Recall that sample/hold circuits are simply analog voltage memory devices. As you are now aware, a sample/hold, or S/H, circuit can be constructed using a standard voltage follower op amp circuit and a capacitor. In addition, monolithic S/H devices are available from several semiconductor manufacturers.

Sample/hold devices are used in data acquisition systems to sample and hold an analog signal while it is being converted by the analog-to-digital converter. In many applications, the analog signal to be converted is changing too fast to be accurately measured by the converter. This problem is illustrated in Figure 6-38. Here, the peak of an analog signal must be measured. However, even the relatively fast successive approximation ADC is too slow to "catch" the peak of a rapidly changing signal. The solution to the problem is to use a sample/hold device to catch and hold the level to be measured. The holding action of the sample/hold device provides sufficient time for accurate measurements by even the slowest analog-to-digital converters.

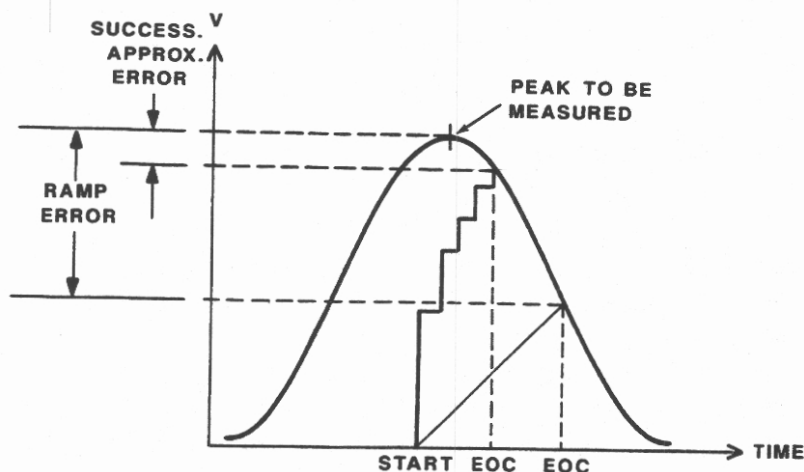


Figure 6-38

Many analog signals change too fast to be accurately measured without using S/H devices.

There are two operating modes of a commercial S/H device - sample and hold. In the sample mode, the output of the device follows its input. However, in its hold mode, the S/H device "freezes" the changing analog input signal. Recall that commercial S/H devices include a logic control line to select between the sample and hold modes. A typical S/H device is shown in Figure 6-39. A given logic level present on the control line puts the device in its sample mode while the opposite logic level puts the device in its hold mode.

Now, suppose you wish to catch the peak of an analog signal for subsequent measurement by an analog-to-digital converter. To do this, the S/H device is first set to its sample mode. Then, when the peak of the analog signal arrives, the device is set to its hold mode.

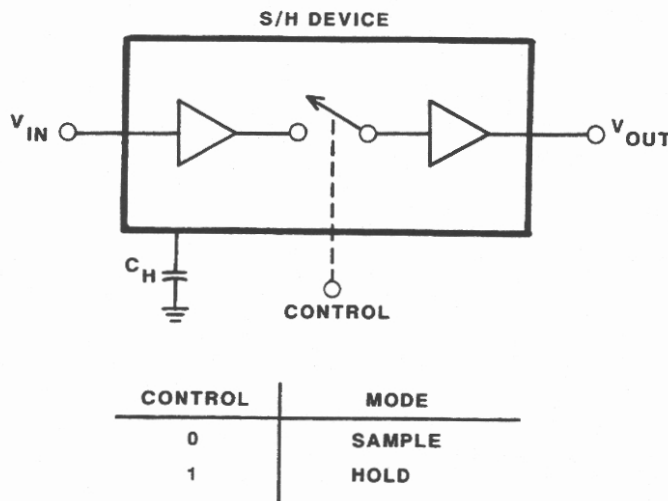


Figure 6-39

A S/H device has a logic input control line to select between the sample and hold modes.

A timing diagram of this process is shown in Figure 6-40. We have assumed that a logic 1 level places the S/H device in its hold mode. The top of Figure 6-40 illustrates the rapidly changing analog input signal to be measured. The output from the S/H device is shown at the bottom of the figure. Notice that, with the proper control signals, the S/H device freezes the peak level of the analog input signal for subsequent measurement by an analog-to-digital converter.

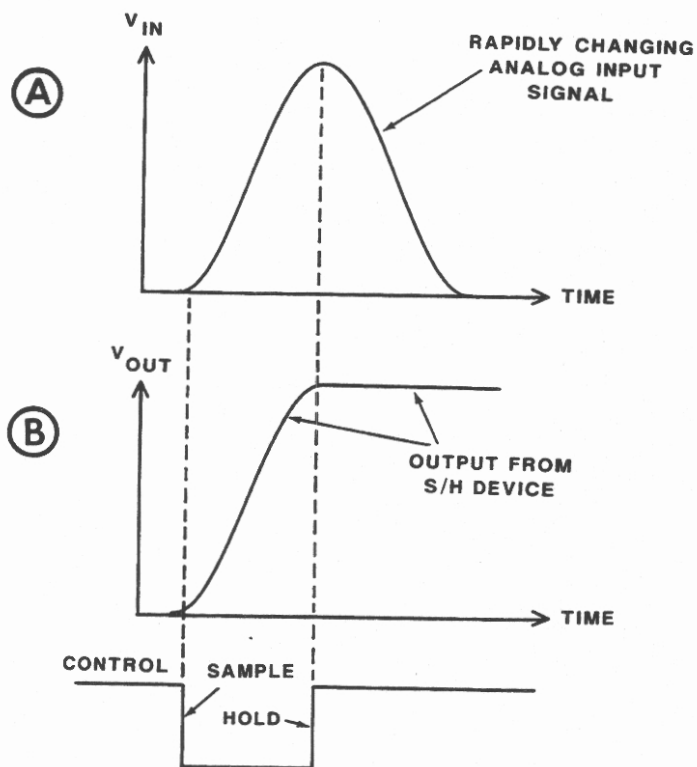


Figure 6-40

Output control of a typical S/H device.

There are generally two ways in which sample/holds are used in data acquisition systems. A data acquisition system that uses a single S/H device is shown in Figure 6-41. Notice that the S/H device is located between the ADC and analog multiplexer. Consequently, external multiplexing must be provided. An ADC with internal multiplexing feature could be used in this circuit, but its internal multiplexing feature could not be used. Here's how it works. The MPU first selects the desired analog input channel by writing the appropriate channel-select logic to the multiplexer through the *P15 - P17* output lines. The analog inputs can be selected sequentially, or at random by the MPU. The MPU then toggles the *P14* output line of the MUART to sample and hold the selected analog input. The *P14* output line of the MUART can control the S/H device directly, or operate the S/H device through control logic. Once the signal is held, the MPU starts the conversion process by reading Port 2 of the MUART. When the conversion is complete, the MPU can read the converted value from Port 2, select another analog input channel, and repeat the process.

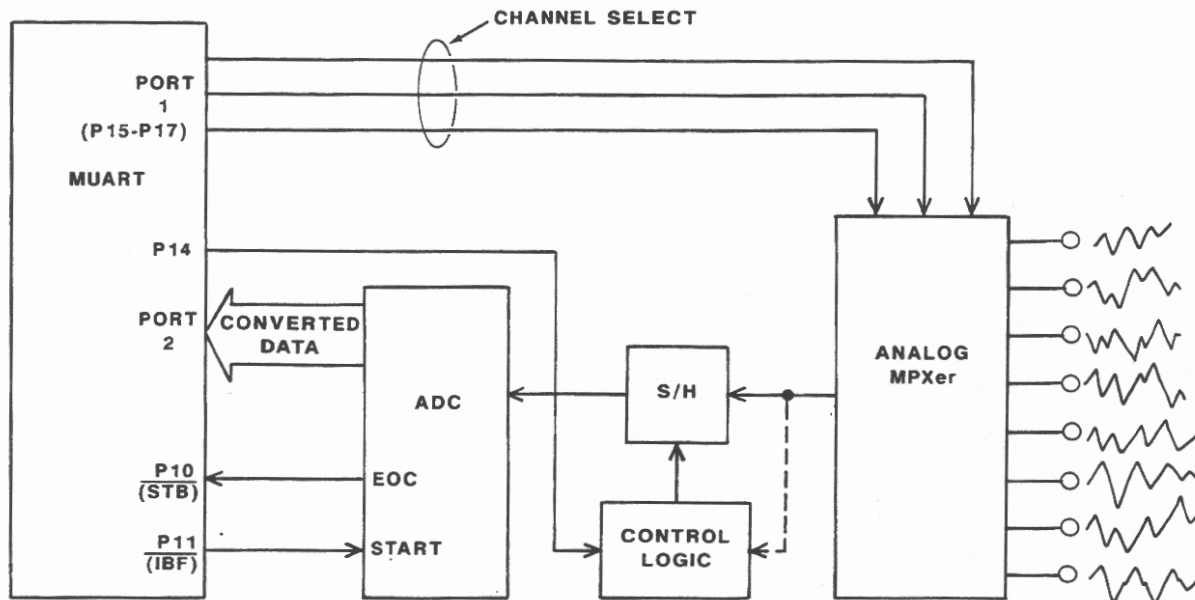


Figure 6-41

A S/H device can be used between the ADC and analog multiplexer in a data acquisition system.

Another way to use S/H devices in a data acquisition system is illustrated in Figure 6-42. Here, a separate S/H device is used for each analog input channel. Notice that the S/H devices are located at the front end of the system - before the analog multiplexer. Consequently, an ADC with internal multiplexing can be fully utilized in this system. With this arrangement, all the S/H devices are switched into their hold mode at the same time. Then, the signals are multiplexed into the ADC one-at-a-time for measurement. This system has the capability of simultaneous measurement of several analog signals because all the input signals are held at the same time. An application where this might be an advantage is in vibration analysis, where several analog quantities must be monitored at the same time.

The two data acquisition systems shown in Figures 6-41 and 6-42 use parallel output ADCs. However, V/F converters can also be used, especially for remote sensing applications. Of course, the V/F converter interface and associated software are different from that of the ADC. However, the analog multiplexer and sample/hold sections of the system will remain the same.

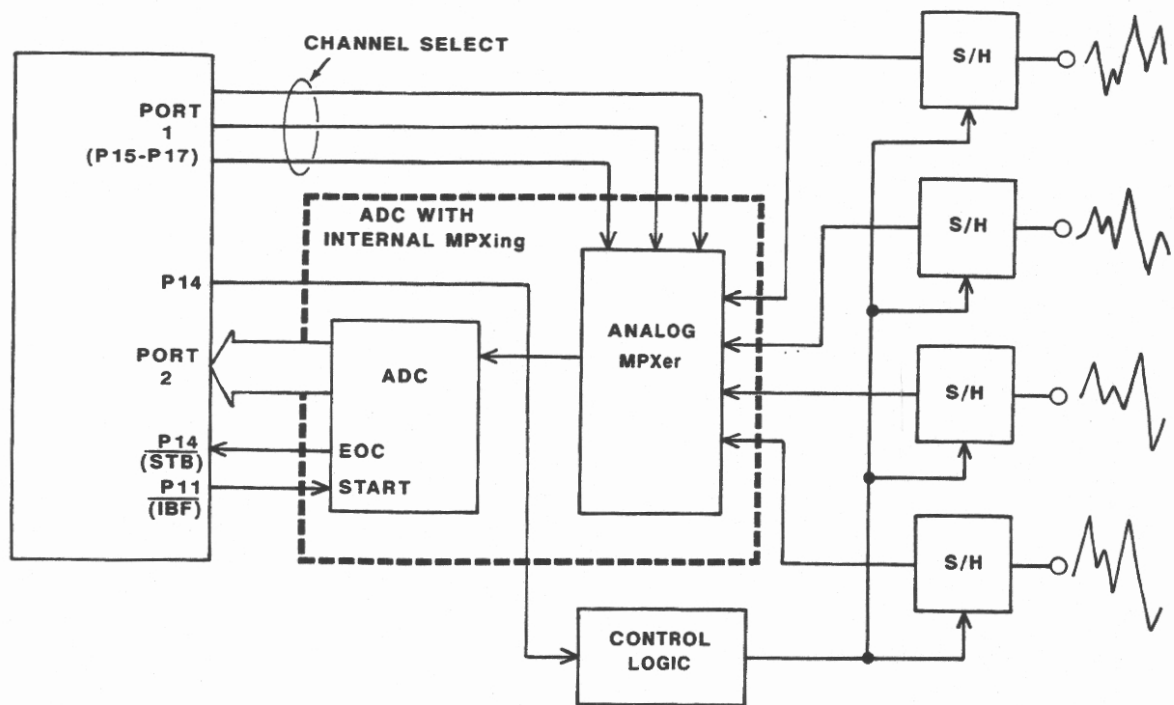


Figure 6-42

Several S/H devices can be used in the front-end of a data acquisition system for simultaneous measurement of analog signals.

## Analog Sensing and Industrial Control Systems

As you are now aware, a microprocessor-based industrial control system must first sense fundamental physical conditions such as temperature, light, magnetic field, or pressure. Analog sensing is provided by a sensor/transducer that converts the physical quantity to be measured into a proportional analog voltage or current. Common sensor/transducer devices include thermistors, photo-transistors, Hall-effect devices, electrical bridges, strain gages and microphones.

The analog signal obtained from the sensor/transducer must be conditioned by a signal conditioner to meet the input requirements of an analog-to-digital converter. Various circuits such as amplifiers, attenuators, comparators, and differential amplifiers are used for analog signal conditioning.

Once the analog signal is conditioned, it can be applied to the input of an analog-to-digital converter, where it is converted to a digital format and input to the MPU via the analog-to-digital converter interface. The MPU must then make a decision based on the sensed condition and output a control word to an actuator which converts a voltage or current into a mechanical action. Examples of actuators include mechanical relays, solid-state relays, solenoids, and speakers.

In summary, a microprocessor-based industrial control system can be divided into three major parts as shown in Figure 6-43. The *analog sensing circuits* include the sensor/transducers, signal conditioners, analog multiplexers, S/H devices, and analog-to-digital converters. The *microprocessor system* consists of the MPU, RAM, ROM, I/O interfaces, and software required for the system. Finally, the *analog control circuits* consist of the digital-to-analog converters, signal amplifiers/buffers, and actuators.

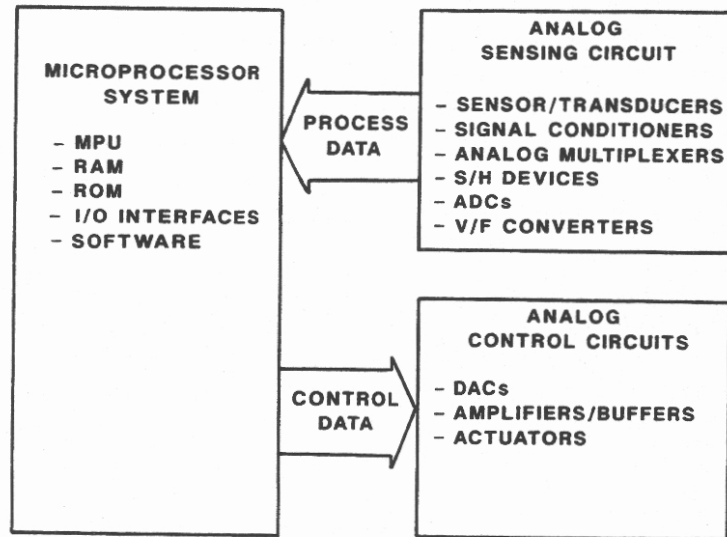


Figure 6-43

A microprocessor-based industrial control system consists of three major parts - the analog sensing circuits, the microprocessor system, and the analog control circuits.

You are now familiar with most of the control system. We will discuss sensor/transducers, signal conditioners, and actuators in detail later. However, at this time, we will briefly discuss some representative analog sensing and control applications to provide you with an introduction to the material in Unit 7.

A simple sense and control application is shown in Figure 6-44. This diagram illustrates a simple way to control the speed of a DC motor. The analog sensor/transducer is a small DC generator that is attached to the shaft of the DC motor to be controlled. As the motor shaft turns, the DC generator produces an output voltage level that is proportional to the speed of the motor. In addition, the polarity of the DC generator output indicates the direction of rotation of the motor shaft.

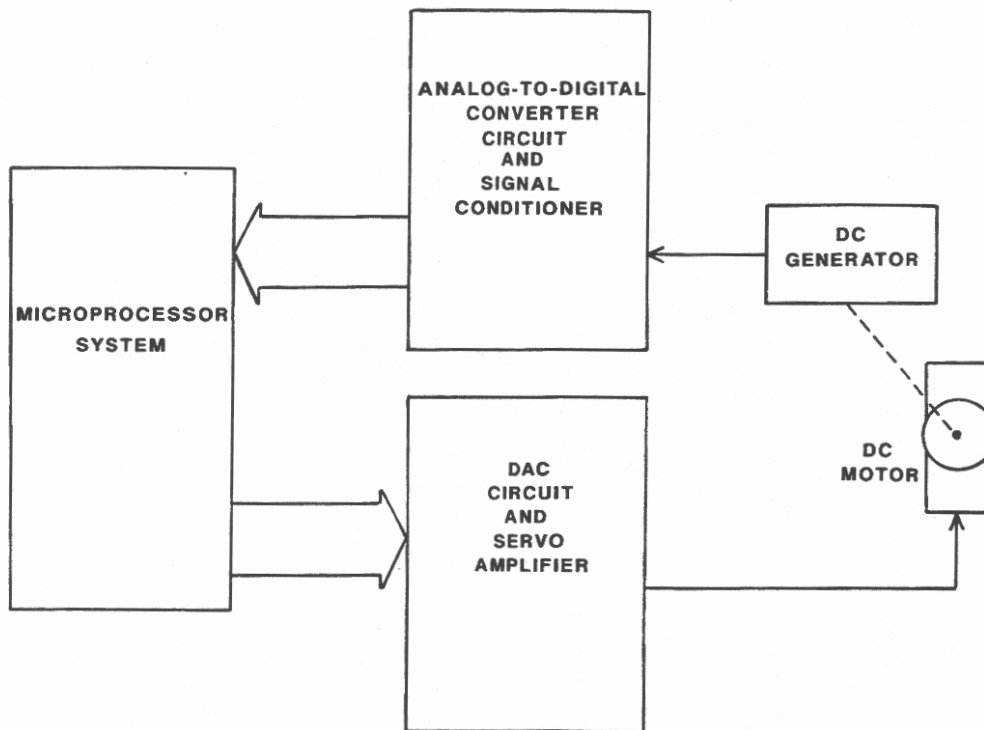


Figure 6-44

Controlling the speed of a DC motor using a closed-loop feedback system.

Once the generator output is conditioned, it can be applied to the input of an analog-to-digital converter. Either a parallel output ADC or V/F converter can be used. However, a V/F converter should be used if the motor is in a remote location.



The output of the analog-to-digital converter is input to the microprocessor system via the converter interface. The MPU receives the actual input RPM data and compares it to a desired value stored in memory. It then computes a correction value and outputs it to the DAC circuit, if a different RPM is required. The DAC circuit converts the new RPM value to a proportional analog control voltage which is amplified by a servo amplifier and applied to the DC motor.

A more complex analog sense and control application is shown in Figure 6-45. In this system, a microprocessor is being used to control a chemical process. The chemicals in the vat must be mixed according to a specific temperature/pressure relationship. The analog sensing circuit consists of two sensor/transducers, a 2-channel analog multiplexer, a single S/H device, and an ADC or V/F converter.

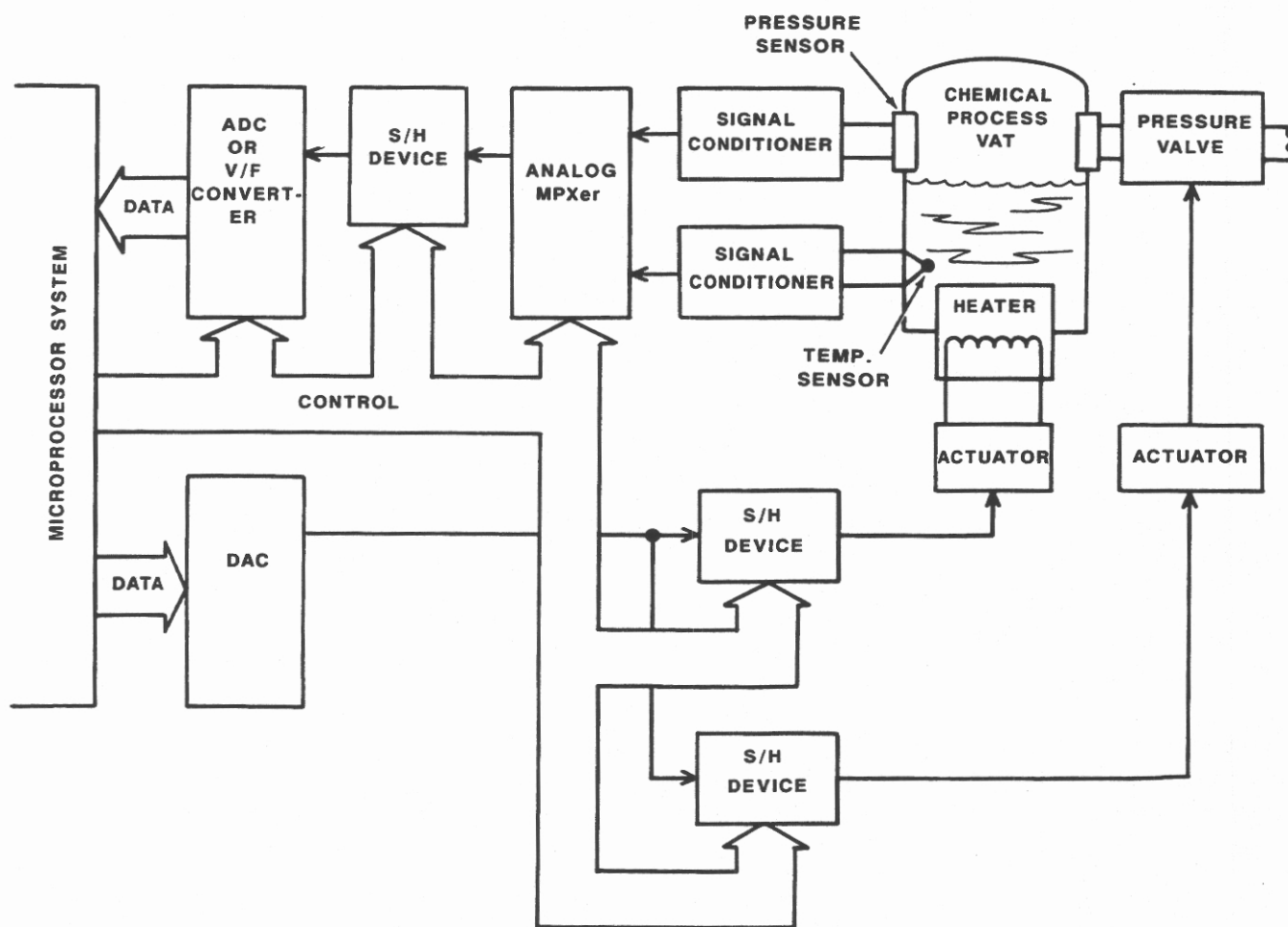


Figure 6-45

Microprocessor control of an industrial chemical process.

The temperature and pressure of the process are each sensed by a transducer connected to the vat. The transducer outputs are conditioned by the signal conditioner circuits and multiplexed into the analog-to-digital converter via the analog multiplexer and S/H device. When the MPU receives the input temperature and pressure data, it must make a control decision based on the desired conditions stored in memory. If a correction is required, the MPU must output control values to the control circuit. The control circuit consists of a single DAC, two S/H devices, and two actuators. Digital control information from the MPU is converted to analog signals by the DAC and directed to the proper actuator via the control logic applied to the S/H devices.

The MPU can be programmed to monitor the process continually to assure very precise control for the critical processes. On the other hand, the MPU could be programmed to monitor the process every few minutes or seconds for less critical processes. If this is the case, the MPU is freed to perform additional control tasks for other processes. It is important to keep time and speed in perspective. The process parameters, like temperature and pressure, might change over many seconds, minutes, or even hours. However, the MPU and its associated components operate in the microsecond range. Consequently, a single microprocessor can be used for several sense and control applications.

Many times the MPU in an industrial control system is used less than 50% of its time for actual sensing and control of the process. The rest of the time it is simply waiting, or wasting time, between sensing and control tasks. It will be your job to make the most efficient use of the MPU's time. This will not be hard to do if you remember that the application always dictates the system configuration and design.

You now have the knowledge to design most of the microprocessor sensing and control system shown in Figure 6-45. However, to complete the system, you must learn more about sense and control circuits. These are the topics in Units 7, 8, and 9.

## Self-Test Review

31. What are the three major application areas of analog-to-digital converters?
32. Define a microprocessor data acquisition system.
33. Why is a S/H device(s) required in many data acquisition systems?
34. What are the two general techniques used to locate S/H devices in a data acquisition system?
35. What is a signal conditioner?
36. Why is an analog multiplexer required in many data acquisition systems?

## Answers

31. The three major application areas for analog-to-digital converters are digital instrumentation, data acquisition, and industrial control systems.
32. A microprocessor data acquisition system is a microprocessor-based system which consists of analog-to-digital converters, analog multiplexers, sample-holds, signal conditioners, and sensor/transducers which process one or more analog signals.
33. An S/H device is required in data acquisition systems to sample and hold a rapidly changing analog signal while it is being converted by the analog-to-digital converter.
34. A single S/H device can be located between the ADC and analog multiplexer, or a separate S/H device for each analog input channel can be used on the front end of the system, before multiplexing.
35. A signal conditioner is usually an electronic circuit that conditions the analog signal received from a sensor/transducer to meet the input requirements of an analog-to-digital converter. Various circuits such as amplifiers, attenuators, comparators, and differential amplifiers are used for analog signal conditioning.
36. To enable a single ADC to measure and convert more than one analog signal.

## UNIT SUMMARY

In this unit, you have learned how to interface D/A and A/D converters to an MPU. In addition, you have learned about several D/A and A/D applications.

The interface between the MPU and a DAC requires an address decoder and a latch. A single MUART output port is employed when interfacing to an 8-bit DAC. However, two MUART ports must be used when interfacing to a DAC of more than eight bits. In addition, an external latch must be provided to prevent a glitch problem.

There are numerous microcomputer applications for D/A converters, especially in the control industry. A microprocessor controlled DAC can be used to generate waveforms, control X-Y displays, control the speed and position of DC motors, and perform various process-related control tasks that require an analog signal. Analog multiplexers and sample/hold devices are required when a single DAC is employed for multiple control tasks.

There are two types of A/D converters: parallel output ADCs and serial output V/F converters. A MUART is used as the interface between a parallel output ADC and the MPU. Due to the relative slow speed of the ADC, the MUART must be configured for a handshake operation to control the conversion process. V/F converters are interfaced to the MPU using hardware or software techniques. A MUART and a binary counter are required for the hardware interface. When a software interface is employed, the V/F converter output is applied to an interrupt input line of the MPU. Software is then written to count the V/F converter pulses over a given time period.

The three major areas of industrial applications for A/D converters is in digital instrumentation, data acquisition, and industrial control systems. The A/D converter is at the heart of most digital instruments. Industrial control systems employ A/D converters to sense real world events. Since most real world events are analog in nature, an A/D converter must be used to convert the analog output of a sensor to a digital value that can be measured by the MPU.

*Unit 7*

**TEMPERATURE AND OPTICAL  
SENSING**

## CONTENTS

|                                                    |      |
|----------------------------------------------------|------|
| Introduction .....                                 | 7-4  |
| Unit Objectives .....                              | 7-5  |
| Temperature Sensing .....                          | 7-6  |
| Thermoresistive Temperature Sensing Devices .....  | 7-6  |
| Resistance Temperature Detectors, or RTDs .....    | 7-7  |
| Construction and Operation .....                   | 7-8  |
| Characteristics .....                              | 7-10 |
| Application Circuits .....                         | 7-12 |
| Thermistors .....                                  | 7-14 |
| Construction And Operation .....                   | 7-14 |
| Characteristics .....                              | 7-16 |
| Application Circuits .....                         | 7-17 |
| Thermoelectric Temperature Sensing Devices .....   | 7-18 |
| Construction and Operation .....                   | 7-19 |
| Characteristics .....                              | 7-20 |
| Application Circuits .....                         | 7-22 |
| Semiconductor Junction Devices .....               | 7-27 |
| Thermoswitches .....                               | 7-31 |
| Construction, Operation, and Characteristics ..... | 7-33 |
| Liquid Expansion Thermoswitches .....              | 7-33 |
| Self-Test Review .....                             | 7-36 |
| Answers .....                                      | 7-38 |
| Optical Sensing .....                              | 7-40 |
| Photoconductive Devices .....                      | 7-40 |
| Construction and Operation .....                   | 7-40 |
| Characteristics .....                              | 7-42 |
| Application Circuits .....                         | 7-44 |
| Photovoltaic Devices and Photodiodes .....         | 7-46 |
| Construction and Operation .....                   | 7-47 |
| The Photovoltaic Mode .....                        | 7-48 |
| The Photoconductive Mode .....                     | 7-50 |
| Characteristics .....                              | 7-51 |
| Application Circuits .....                         | 7-51 |
| Phototransistors .....                             | 7-53 |
| Construction and Operation .....                   | 7-53 |
| Characteristics .....                              | 7-55 |
| Application Circuits .....                         | 7-55 |

---

|                                         |      |
|-----------------------------------------|------|
| Integrated Optical Sensing Devices..... | 7-57 |
| Construction and Operation.....         | 7-57 |
| Characteristics.....                    | 7-58 |
| Application Circuits.....               | 7-59 |
| Self-Test Review.....                   | 7-64 |
| Answers.....                            | 7-66 |
| Unit Summary.....                       | 7-68 |



## ***Unit 7***

# **TEMPERATURE AND OPTICAL SENSING**

## **INTRODUCTION**

You are about to begin a comprehensive study of the most important, and sometimes the weakest, link in a microprocessor applications circuit — sensors, transducers, and detectors. A sensor, transducer, or detector is required to convert a process variable into an electrical quantity that can be measured by a microprocessor. The microprocessor is then able to make a decision and control the process according to the sensed condition.

As you have already probably noticed, we have divided our discussion of sensors, transducers, and detectors into two units. This is because of the amount of material that needs to be presented to provide you with a working knowledge of these devices. In this unit, you will learn about temperature and optical sensing. Then, Unit 8 is devoted entirely to those devices that are used to measure mechanical type quantities such as position, pressure, force, and flow.

## UNIT OBJECTIVES

1. Explain the operating principles of RTD, thermistor, thermocouple, and semiconductor type temperature sensors.
2. Construct a microprocessor-controlled thermometer.
3. Calculate the resistance of RTDs and thermistors from given temperature coefficient and sensitivity values.
4. Explain the operating principles of photoresistive, photovoltaic, and photoemissive devices.
5. List advantages and disadvantages of several different types of temperature and optical sensors.
6. Compare the relative application merits of photodiodes and phototransistors.
7. State several sensing applications for optical interrupter and optical reflector devices.
8. Explain why optocouplers are used for electrical isolation.

## TEMPERATURE SENSING

Let's begin this unit with temperature sensing, since it is probably the most common physical quantity measured or detected in industrial control systems. You're probably aware that there are three different scales used to measure temperature — Celsius, Fahrenheit, and Kelvin. In this course, you will use the Celsius, (or Centigrade) scale since most scientific instruments and industrial instrumentation are calibrated in °C.

A temperature sensing device is one that converts a temperature into a proportional analog signal. This signal is then conditioned, converted to digital, and used by a digital control system to monitor a given process. In general, there are four major classifications of temperature sensing devices. They are: **thermorestistive devices**, **thermoelectric devices**, **semiconductor junction devices**, and **thermoswitches**. Each type of device operates on a different principle and has its own advantages and disadvantages. In most cases, your temperature measurement application will dictate which type of device to use. Now let's begin your study of temperature sensing with thermoresistive devices.

### Thermoresistive Temperature Sensing Devices

Thermoresistive devices operate on the principle that the resistance of a conductor or semiconductor changes with temperature. The resistance change of conductors, such as nickel and platinum, is very predictable and linear for changes in temperature. The resistance change of semiconductors is large for a given temperature change, but nonlinear. Thermoresistive temperature sensors that employ conductive sensing elements are called **resistance temperature detectors**, or **RTDs**, while those that employ semiconductor elements are called **thermistors**.

## RESISTANCE TEMPERATURE DETECTORS, OR RTDs

The relationship between temperature and the resistance of a conductor is called the **temperature coefficient of resistance**, or  $\alpha$  (Greek letter alpha), of the conductor. The temperature coefficients of several common conductors are listed in Table 7-1.

| Temp. Coef.( $\alpha$ ) |        |
|-------------------------|--------|
| Conductor               | Per °C |
| Nickel                  | .0067  |
| Platinum                | .0039  |
| Copper                  | .0038  |
| Tungsten                | .0045  |
| Carbon                  | -.0005 |

Table 7-1  
Temperature Coefficients of Common Conductors.

As you can see, the temperature coefficient of most conductors is a positive value. This means that the conductor resistance changes directly with temperature. As the temperature goes up, so does the resistance. One exception to this rule is carbon, which has a negative temperature coefficient. As a result, the resistance of a carbon conductor changes inversely with temperature. As one factor goes up, the other goes down.

By knowing the resistance of a conductor at one temperature, you can find the resistance of the conductor at a new, or different temperature using its temperature coefficient as follows:

$$R_{\text{New}} = R_{\text{Old}} [1 + \alpha (T_{\text{New}} - T_{\text{Old}})]$$

Where:  $R_{\text{new}}$  is the resistance of the conductor at the new temperature.

$R_{\text{old}}$  is the resistance of the conductor at the old temperature.

$\alpha$  is the temperature coefficient of the conductor.

$T_{\text{new}}$  is the new temperature.

$T_{\text{old}}$  is the old temperature.

**Example 7-1:**

A platinum element RTD has a resistance of 100 ohms at 0°C. Calculate the RTD resistance at a temperature of 50°C.

**Solution:**

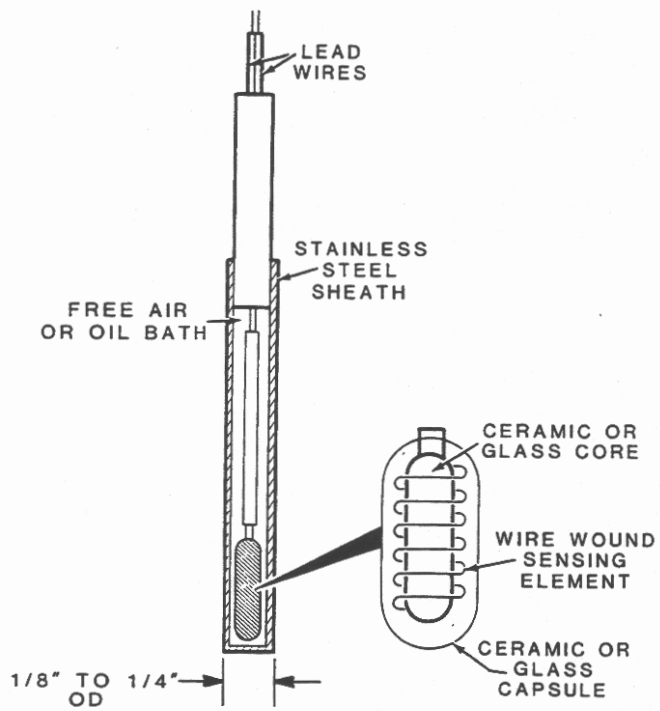
From Table 7-1 you find that the temperature coefficient of platinum is .0039. Using this value in the above relationship you get:

$$\begin{aligned} R_{\text{new}} &= R_{\text{old}} [1 + \alpha (T_{\text{new}} - T_{\text{old}})] \\ &= 100 \, \Omega [1 + .0039 (50^\circ\text{C} - 0^\circ\text{C})] \\ &= 100 \, \Omega [1 + .195] \\ &= 119.5 \, \text{ohms} \end{aligned}$$

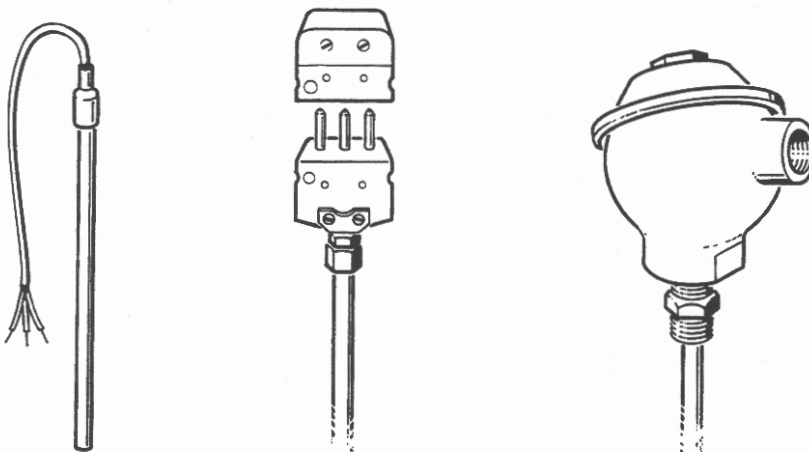
**Construction and Operation**

The construction of a typical wire wound RTD **probe** is shown in Figure 7-1. The probe consists of the temperature sensing element, a sheath, and lead wire termination. The nickel or platinum sensing element is wound around a ceramic or glass core, and sealed within a ceramic or glass capsule. The capsule is then inserted into a hollow metal sheath or closed-end tube. The sheath protects the fragile sensing element from moisture, and the hostile environment. The sheath is usually constructed of stainless steel and has a diameter of 1/8" to 1/4". Its length can range anywhere from three inches to three feet.

The lead wire termination consists of two, three, or four lead wires that exit the probe. The most common termination consists of three wires for connection into a Wheatstone bridge circuit. This connection will be discussed shortly. The wires exit the probe through a variety of configurations as shown in Figure 7-1B. Observe that one type of arrangement provides for a quick-disconnect of the probe. Another provides a connector head that can be threaded into a container.



(A)



(B)

Figure 7-1

- A. The construction of a typical wire wound RTD probe.  
B. Various probe configurations.

Another type of RTD is the thin film detector, or TFD, shown in Figure 7-2. This device is constructed by depositing a thin film of platinum on a flat ceramic substrate. The thin film device is then encapsulated in silicone rubber to provide a moisture proof seal. The advantages of this type of construction are its size, accuracy, and fast response time. TFDs are about the size of an ordinary small signal transistor. Their resistance tolerance is less than .1% and they respond to temperature changes in less than .5 seconds.



ACTUAL SIZE

Figure 7-2

A thin film detector, or TFD, is smaller, more accurate, and faster responding than a wire wound RTD.

## Characteristics

The most common conductor elements used in the construction of RTDs are nickel and platinum. You can see why, by looking at the resistance versus temperature curves in Figure 7-3. Notice that both curves are extremely linear, especially the platinum curve. Platinum RTDs are used for their extreme accuracy, linearity, and broad temperature range. Nickel RTDs are used for their relative low cost and sensitivity. Observe that the nickel curve is steeper than the platinum curve. Consequently, small changes in temperature result in larger resistance changes for nickel RTDs. Nickel RTDs are used to measure temperatures from  $-180^{\circ}$  to  $300^{\circ}\text{C}$ , while platinum RTDs have a range from  $-200^{\circ}\text{C}$  to  $600^{\circ}\text{C}$ . Table 7-2 lists several resistance-versus-temperature values for both platinum and nickel RTDs.

| TEMPERATURE | PLATINUM RTD | NICKEL RTD |
|-------------|--------------|------------|
| -50°C       | 80 Ohms      | 80 Ohms    |
| -25°C       | 90 Ohms      | 100 Ohms   |
| 0°C         | 100° Ohms    | 120 Ohms   |
| 25°C        | 110 Ohms     | 140 Ohms   |
| 50°C        | 120 Ohms     | 160 Ohms   |
| 100°C       | 140 Ohms     | 200 Ohms   |
| 200°C       | 180 Ohms     | 280 Ohms   |
| 300°C       | 220 Ohms     | 360 Ohms   |

Table 7-2

Approximate resistance-versus-temperature values  
of typical platinum and nickel RTDs.

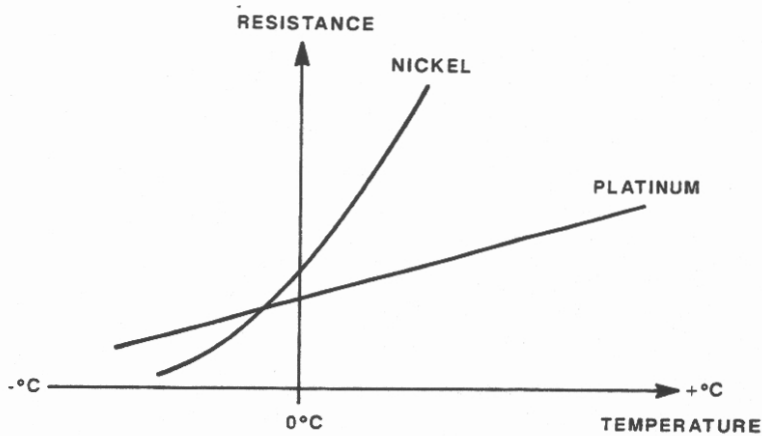


Figure 7-3

Resistance-versus-temperature curves for nickel and platinum RTDs.



## Application Circuits

An RTD can be connected as part of a potentiometer circuit as shown in Figure 7-4. Here, a two-wire RTD is connected in series with a variable calibration resistance,  $R_{cal}$ . The output voltage of the potentiometer circuit is taken across the RTD and can be applied directly to the input of an A/D converter, provided that it meets the input signal requirements of the A/D converter. If not, an intermediate signal conditioning circuit can be added to strengthen or weaken the signal.

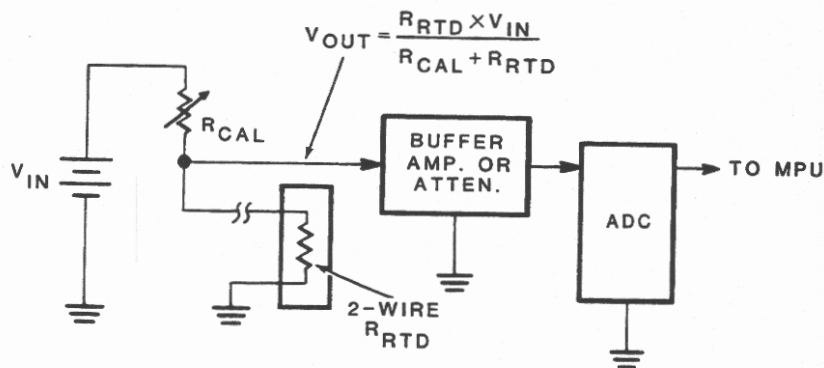


Figure 7-4

A 2-wire RTD in a potentiometer circuit.

In Figure 7-5, the RTD forms one leg of a DC Wheatstone bridge circuit. Naturally, the bridge circuit and RTD must be separated by some distance so that the temperature being measured does not affect the other bridge resistances. However, this creates a problem — as the RTD lead wires become longer, their resistance increases to a point where they affect the temperature reading. The solution to this problem is to use a three-wire RTD as shown to compensate for lead resistance. Observe that the RTD is connected so that lead wire A is in the  $R_{cal}$  leg of the bridge. Lead wire C is in the  $R_{cal}$  leg of the bridge. As long as  $R_{cal}$  is close to  $R_{RTD}$ , the effects of the lead resistances are negligible. If lead wires A and C have the same resistance, their effects on the bridge output will cancel each other, since they are located in adjacent legs of the bridge. The middle lead wire B, has no effect on the bridge output since it doesn't carry any current when the bridge is balanced.

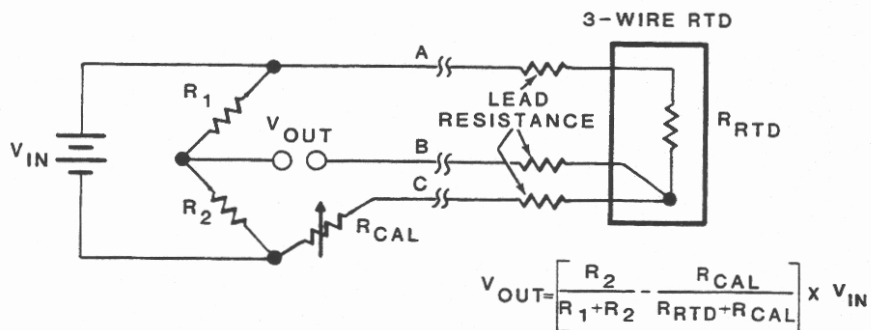


Figure 7-5

A 3-wire RTD in a DC wheatstone bridge circuit.

The bridge circuit connection will produce an output voltage that is proportional to the resistance of the RTD. The variable calibration resistor,  $R_{cal}$ , can be used to calibrate or balance the bridge output. Recall that the output of a Wheatstone bridge is a *differential output*, meaning that it is not ground referenced. However, most A/D converters require a ground referenced input. An op amp signal conditioning circuit called an *instrumentation amplifier* is often required to make the bridge output compatible with the input of an A/D converter.

### Example 7-2:

A platinum RTD is placed in a 12V DC Wheatstone bridge shown in Figure 7-5. Suppose the resistance of each of the RTD leads is 50 ohms, and that  $R_1$  and  $R_2$  both have a resistance of 10 K $\Omega$ . Also, suppose the RTD exhibits a resistance change due to temperature, per Table 7-2.

- A. To what value must the calibration resistor,  $R_{cal}$ , be adjusted in order to balance the bridge output at 0°C?
- B. Assume the bridge output is balanced at 0°C, what will the output be at room temperature, or 25°C?

### Solution:

- A. Since a three-wire RTD is being used, the effect of lead resistance is cancelled-out and does not have to be considered. Now, recall from DC circuits that the bridge is nulled, or balanced when:

$$\frac{R_1}{R_2} = \frac{R_{RTD}}{R_{cal}}$$

Since  $R_1$  and  $R_2$  are equal, the above equation dictates that  $R_{cal}$  must equal the RTD resistance to balance the bridge. From Table 7-2, the RTD has a resistance of 100 ohms at 0°C. Therefore, the calibration resistance must be adjusted to 100 ohms to balance the bridge when the temperature is 0°C. Of course, when the bridge is nulled, or balanced, its output is 0 volts.

- B. According to Table 7-2, the resistance of the RTD is 110 ohms at 25°C. From DC circuits you might recall that the output of the Wheatstone bridge in Figure 7-5 is:

$$V_{out} = \left[ \frac{R_2}{R_1 + R_2} - \frac{R_{cal}}{R_{RTD} + R_{cal}} \right] \times V_{in}$$

Our circuit has the following values:

$$\begin{aligned}V_{in} &= 12 \text{ v} \\R_1 &= 10 \text{ k}\Omega \\R_2 &= 10 \text{ k}\Omega \\R_{\text{RTD}} &= 110 \Omega \text{ at } 25^\circ\text{C} \\R_{\text{cal}} &= 100 \Omega \text{ (part a)}\end{aligned}$$

Using these values in the above relationship, you get:

$$\begin{aligned}V_{\text{out}} &= \left[ \frac{10 \text{ k}\Omega}{10 \text{ k}\Omega + 10 \text{ k}\Omega} - \frac{100 \Omega}{110 \Omega + 100 \Omega} \right] \times 12 \text{ V} \\&= [.5 - .476] \times 12 \text{ V} \\&= .288 \text{ V}\end{aligned}$$

You should be aware that any current passing through an RTD creates a **self-heating** effect, due to the  $I^2R$  power dissipation of the resistance element. The self-heating effect results in the RTD temperature being higher than the actual temperature. To avoid extensive self-heating, the current through the RTD must be kept to a minimum. The RTD manufacturer will usually specify a self-heating value called the **dissipation constant**. The dissipation constant of an RTD is the power required to raise the temperature of the RTD element by  $1^\circ\text{C}$ . For instance, a dissipation constant of  $100 \text{ mw}/^\circ\text{C}$  means that the temperature of the RTD is raised by  $1^\circ\text{C}$  for every 100 milliwatts of power loss within the RTD, due to self-heating. You must compensate for this self-heating, in the signal conditioning circuit, or the MPU software.

## THERMISTORS

A thermistor is another thermoresistive device whose resistance changes with temperature, like an RTD. However, a thermistor is constructed from semiconductor material, resulting in a device that exhibits characteristics quite different from an RTD.

### Construction And Operation

Thermistors are usually manufactured from oxide compounds of elements such as cobalt, copper, magnesium, iron, manganese and nickel, just to mention a few. The particular type of semiconductor used depends on the desired resistance range, temperature range, and sensitivity. In addition, the doping level of the semiconductor can be varied to obtain different operating characteristics.

The construction of a typical thermistor probe is shown in Figure 7-6. The sensing element consists of a small bead of semiconductor that is molded around a pair of leads. The bead is then often sealed in a teflon or glass tube as shown, to protect it from moisture and other environmental hazards.

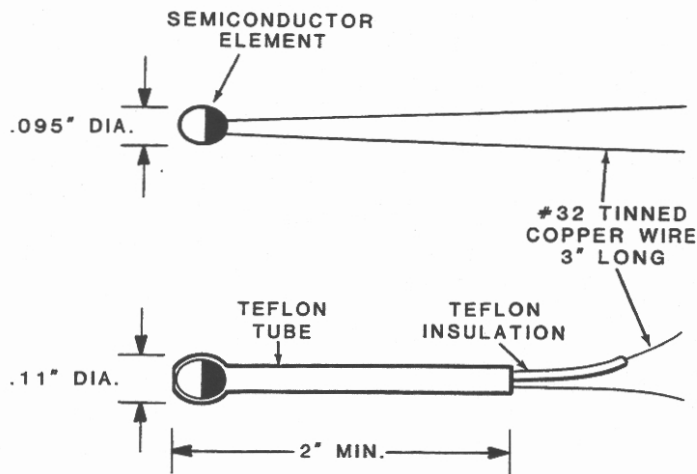


Figure 7-6

The construction of a typical bead type thermistor.

Several different thermistor sensors are illustrated in Figure 7-7, along with the electronic symbol for the thermistor. As you can see, several sizes and shapes are available to meet different applications. The various shapes are made possible by the molding ability of the semiconductor sensing element.

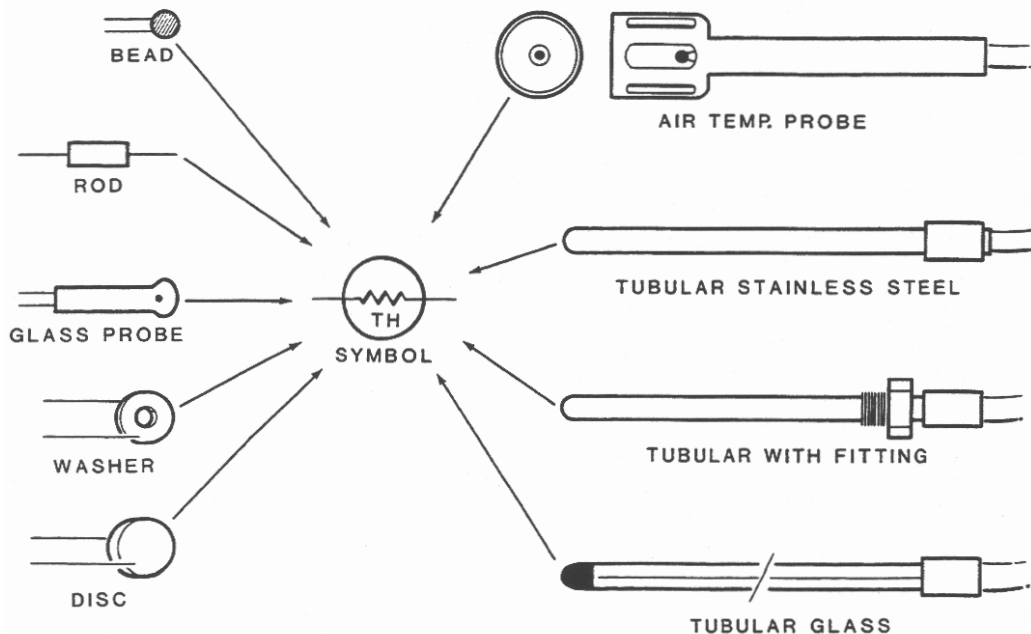


Figure 7-7

Thermistors are manufactured in many different forms to "fit" the applications.

## Characteristics

A thermistor is basically a temperature sensitive resistor made from semiconductor material. As you can see from Figure 7-8, the resistance of a thermistor changes inversely with temperature. In other words, its resistance decreases with increases in temperature.

Looking at the curve in Figure 7-8 you should observe two common characteristics of a thermistor. First, it is extremely sensitive. In fact, the thermistor is by far the most sensitive of all temperature sensing devices. Since its change in resistance is so great, with a corresponding temperature change, the thermistor can detect even slight changes in temperature. This is a trait other temperature sensing devices cannot claim.

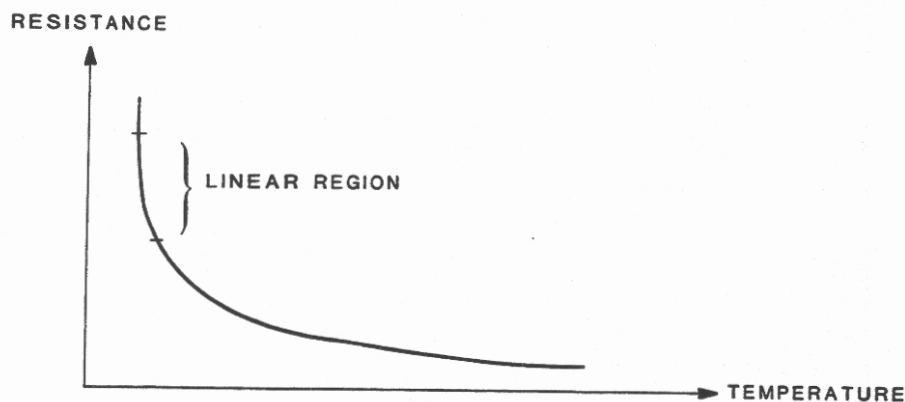


Figure 7-8

Resistance-versus-temperature curve for a typical thermistor.  
Note, that resistance decreases nonlinearly with increases in temperature.

The sensitivity of a thermistor is usually specified as a percent of its nominal resistance. For example, if a thermistor has a nominal resistance of 10 k $\Omega$  at room temperature, or 25°C, and a sensitivity of 3%/°C, its resistance changes approximately 300 ohms for each 1°C of temperature change. At 0°C, the resistance of the thermistor is 17,500 ohms, and at 50°C its resistance is 2,500 ohms. Such sensitivity allows the thermistor to be conveniently used in potentiometer and bridge circuits to provide temperature sensing to within 1°C. In many cases, thermistors can be used to detect temperature changes as small as .1°. For this reason, they are often used as the sensing element in electronic thermometers. Table 7-3 lists several resistance-versus-temperature values for a typical thermistor.

| TEMPERATURE | RESISTANCE |
|-------------|------------|
| -30°C       | 16 K       |
| -10°C       | 13 K       |
| 0°C         | 12 K       |
| 10°C        | 11 K       |
| 30°C        | 8 K        |
| 50°C        | 6 K        |

Table 7-3

Resistance-Versus-Temperature for a typical thermistor.

Because thermistors are made from semiconductor material, they are not reliable for measuring temperatures above 100°C. Exposure to higher temperatures for long periods of time will cause the thermistor to drift from its specified resistance ratings.

The second thing to note from the characteristic curve in Figure 7-8 is the extreme non-linearity of the thermistor. This is the price you must pay for increased sensitivity. Before microprocessors, non-linearity was a severe disadvantage, since analog linearization circuits were required to make the output of a thermistor linear. Now, with microprocessors, software look-up tables eliminate any need for special linearization circuits. If you are not familiar with software look-up tables, you will see how they are used when you perform the experiments in this course.

The response time of a thermistor depends on its size. Small thermistors have response times of .1 seconds or less, while larger thermistors might require over 10 seconds to stabilize. Finally, since thermistors are resistive devices, like RTDs, a certain amount of self-heating occurs. Consequently, current levels through the device must be kept to a minimum. Typical dissipation constants for thermistors range from 1 mW/°C to 10 mW/°C.

## Application Circuits

Like the RTD, the thermistor is generally used as part of a DC Wheatstone bridge circuit as shown in Figure 7-9A. The major difference is that a thermistor does not require any compensation for lead resistance. This is due to the relatively large thermistor resistance as compared to any lead resistance. In other words, the lead resistance is so small as compared to the thermistor resistance that it can be neglected for all practical purposes.

Another application circuit is shown in Figure 7-9B. Here, the thermistor is one of the feedback resistors in a **non-inverting op amp** circuit. As you can see from the associated equation, the voltage output,  $V_{out}$ , of the circuit is controlled directly by the resistor values. If  $R_{cal}$  is held constant,  $V_{out}$  is directly proportional to the thermistor resistance value,  $R_{TH}$ .

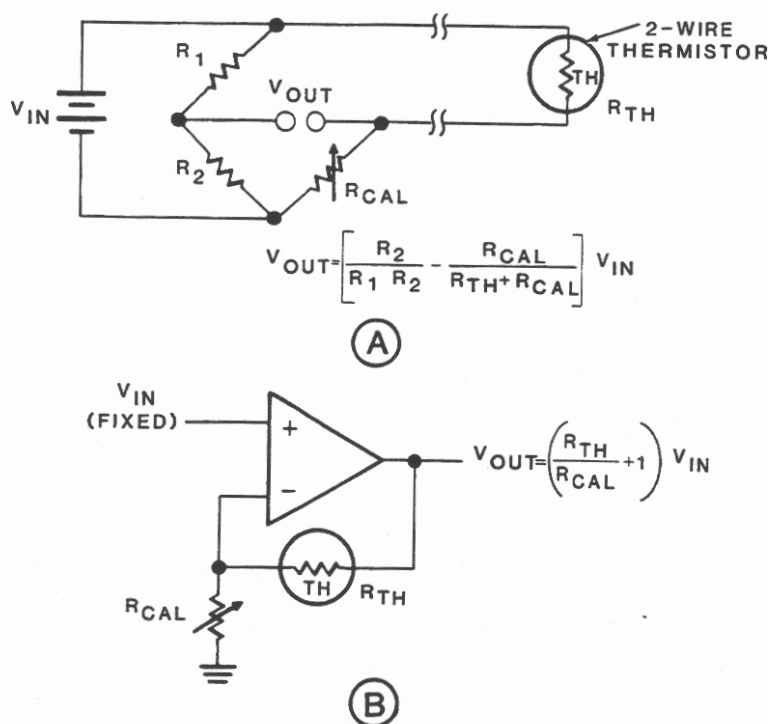


Figure 7-9

- A. A 2-wire thermistor in a DC wheatstone bridge.  
B. A non-inverting op amp circuit.

## Thermoelectric Temperature Sensing Devices

Up to this point, all the temperature sensors discussed have operated on the principle of a changing resistance with temperature. Another category of sensors, called **thermocouples**, generate a changing voltage as a function of temperature. The thermocouple can therefore be classified as a thermoelectric transducer, since it converts thermal energy into electrical energy.

## CONSTRUCTION AND OPERATION

A thermocouple is made by soldering, or welding, two dissimilar metal wires together as shown in Figure 7-10. The joined end is called the *hot end* and the open end is called the *cold end*. When the hot end is heated as shown, a potential difference, or voltage, is generated across the cold, or open end of the thermocouple. The reason for this is that one wire will give up electrons faster than the other wire, since they are made of different materials. The wire that gives up electrons the fastest, exhibits a positive charge relative to the charge on the other wire. As a result, a potential difference is created across the junction that is proportional to the temperature of the junction. The higher the temperature at the junction, the higher the potential difference across the junction. This potential difference is called a *Seebeck voltage*, after Thomas Seebeck who discovered the thermoelectric effect in 1821.

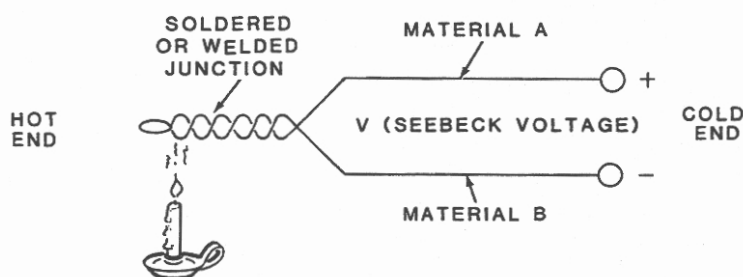


Figure 7-10

A thermocouple is made by joining two dissimilar metals, or wires, at one end.

Commercial thermocouples are available in all shapes and sizes to meet just about any application. A few of the more common types are pictured in Figure 7-11. Notice that you can obtain bare unsheathed thermocouples, as well as various mounting assemblies to meet a particular measurement application. Many times the thermocouple is enclosed like an RTD in a stainless steel sheath-type probe, to protect it from harsh environments.

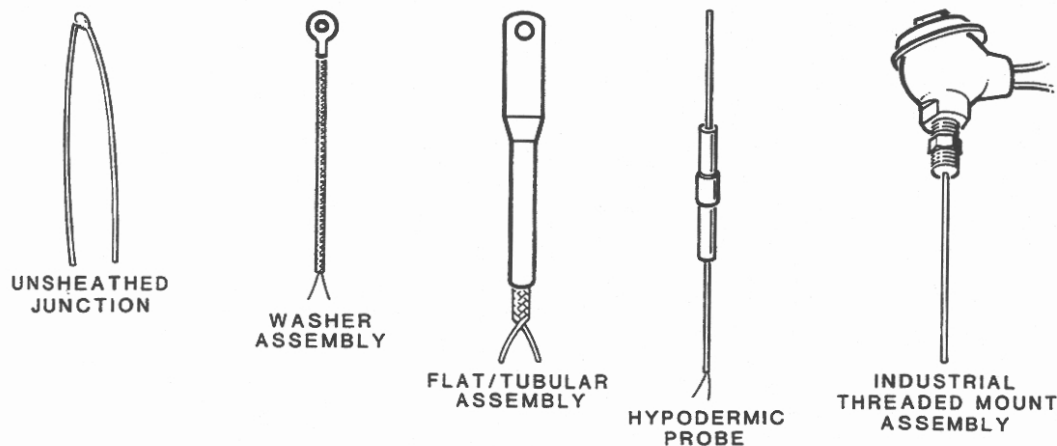


Figure 7-11

Various types of commercially available thermocouple assemblies.



## CHARACTERISTICS

Thermocouples are made of different material combinations for different temperature ranges. Several common types of thermocouples are listed in Table 7-4, each has a unique alphabetic designation. For example, a type J thermocouple is made of iron and constantan wires and has a temperature range of  $-190^{\circ}\text{C}$  to  $760^{\circ}\text{C}$ . The material listed on the left of the hyphen is the electrically positive material, and the material listed to the right of the hyphen is the electrically negative material.

As Table 7-4 shows, the single biggest advantage of thermocouple sensors over other temperature sensing devices is their temperature sensing range. Note that there is a thermocouple available to meet just about any temperature sensing application. Another point to note from the table is the very small voltage change for a given thermocouple over its range. This is probably the biggest disadvantage of a thermocouple, since such a small voltage is extremely sensitive to noise.

| TYPE | JUNCTION MATERIALS        | TEMP RANGE                                       | VOLTAGE CHANGE OVER ENTIRE RANGE |
|------|---------------------------|--------------------------------------------------|----------------------------------|
| J    | Iron-Constantan           | $-190^{\circ}\text{C}$ to $760^{\circ}\text{C}$  | 50 mV                            |
| K    | Chromel-Alumel            | $-270^{\circ}\text{C}$ to $1260^{\circ}\text{C}$ | 56 mV                            |
| T    | Copper-Constantan         | $-270^{\circ}\text{C}$ to $400^{\circ}\text{C}$  | 26 mV                            |
| E    | Chromel-Constantan        | $-330^{\circ}\text{C}$ to $1600^{\circ}\text{C}$ | 74 mV                            |
| R    | Platinum/Rhodium-Platinum | $0^{\circ}\text{C}$ to $1760^{\circ}\text{C}$    | 19 mV                            |

Table 7-4

A partial list of thermocouple types and associated temperature ranges.

A graph of output voltage versus temperature for the above thermocouples is provided in Figure 7-12. This graph shows that thermocouples are relatively linear devices. Furthermore, the slope of the graph indicates a given thermocouple's sensitivity. For instance, you could say that a type E thermocouple is more sensitive than a type R thermocouple. Again, the application will dictate what type of thermocouple to use. Once a thermocouple type is selected for a given application, you must consult the manufacturers specifications for exact voltage-versus-temperature relationships and operating characteristics.

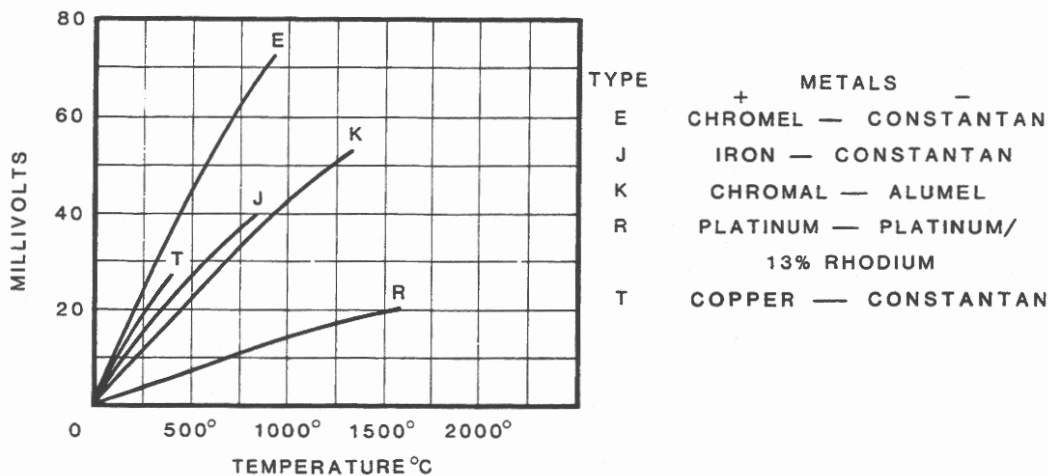


Figure 7-12

Output voltage-versus-temperature, for several common types of thermocouples.

## APPLICATION CIRCUITS

It would be nice if you could simply amplify the thermocouple output voltage, convert it to digital with an A/D converter, and use a computer look-up table to determine the temperature. However, there is one major problem with this measurement process — when the open end of the thermocouple is connected to a signal conditioning circuit, two more thermocouples are created at the connection as shown in Figure 7-13A. The reason for this is that the thermocouple wires are made from a different material than the copper circuit wires. For instance, the type J thermocouple shown is made from iron and constantan. These two wires create two additional thermocouples with the copper circuit leads. As a result, the measured voltage,  $V$ , is not the actual voltage generated by the thermocouple sensor.

The junction created by connecting the thermocouple to the signal conditioning circuit is called a **reference junction**. The voltage seen by the signal conditioning circuit depends on the difference between the temperature at this reference junction,  $T_{ref}$ , and the actual junction temperature,  $T$ . A schematic diagram summarizing this idea is shown in Figure 7-13B.

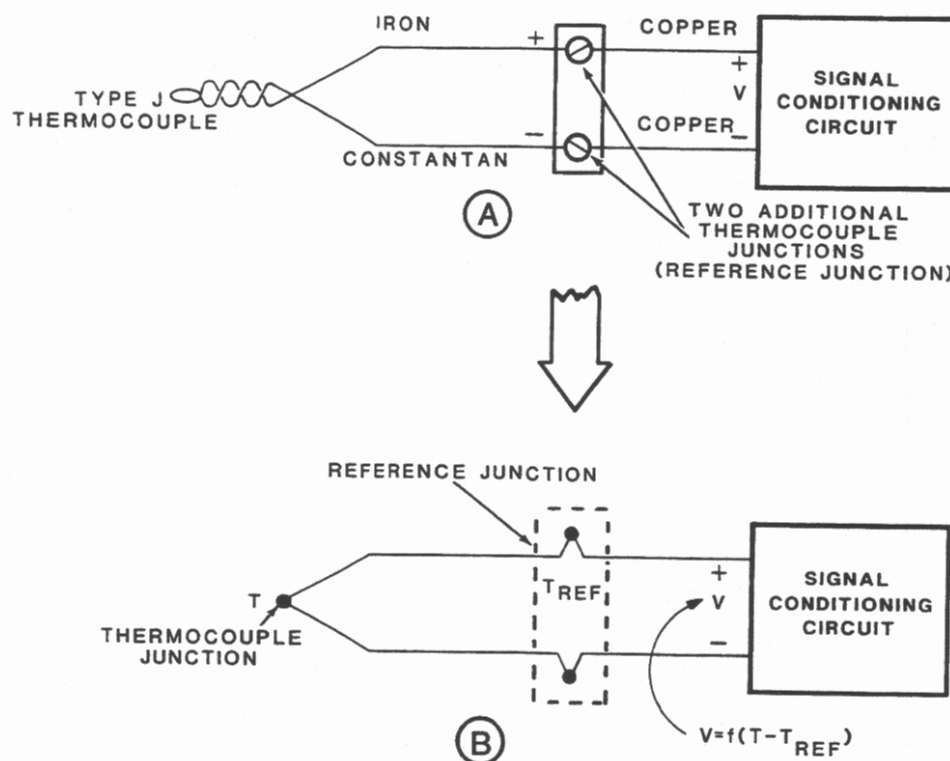


Figure 7-13

A reference junction is created when the thermocouple is connected to another circuit, in this case a signal conditioning circuit.

Now, let's look at Table 7-5. This table provides a partial listing of voltage-versus-temperature, for a type J thermocouple. The table provides voltages for temperatures from  $-150^{\circ}\text{C}$  to  $700^{\circ}\text{C}$  in  $5^{\circ}$  increments. In addition, the table assumes that the reference temperature is  $0^{\circ}\text{C}$ .

**Example 7-3:**

Suppose you are using a type J thermocouple to measure temperature, and the reference temperature is  $0^{\circ}\text{C}$ .

- A. What voltage would be seen by a signal conditioning circuit for a thermocouple temperature of  $125^{\circ}\text{C}$ ?
- B. What temperature is represented by a voltage of 13.56 mV?

**Solution:**

- A. The rows in Table 7-5 provide voltages for each  $5^{\circ}$  increment in temperature. The columns provide voltages for every  $50^{\circ}$  of temperature. To find the voltage corresponding to  $125^{\circ}\text{C}$ , you find the  $100^{\circ}$  row and read across to the  $25^{\circ}$  column. The resulting voltage is 6.63 mV.
- B. A voltage of 13.56 mV is found in the table at the intersection of the  $250^{\circ}$  row and the  $0^{\circ}\text{C}$  column. Thus, the measured temperature is  $250^{\circ}\text{C}$ .

At this point, you might suggest that the voltage-versus-temperature table for a given thermocouple be stored in computer memory and used to "look-up" the temperature for a given voltage. However, remember that Table 7-5 assumes that the reference temperature is  $0^{\circ}$ . This means you must maintain the thermocouple/signal conditioning circuit connections at precisely  $0^{\circ}\text{C}$ . This is highly impractical in an industrial environment.

|    |       | °C →  |       |       |       |       |       |       |       |       |       |
|----|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
|    | T(°C) | 0     | 5     | 10    | 15    | 20    | 25    | 30    | 35    | 40    | 45    |
|    | -150  | -6.50 | -6.66 | -6.82 | -6.97 | -7.12 | -7.27 | -7.40 | -7.54 | -7.66 | -7.78 |
|    | -100  | -4.63 | -4.83 | -5.03 | -5.23 | -5.42 | -5.61 | -5.80 | -5.98 | -6.16 | -6.33 |
|    | -50   | -2.43 | -2.66 | -2.89 | -3.12 | -3.34 | -3.56 | -3.78 | -4.00 | -4.21 | -4.42 |
|    | - 0   | 0.00  | -0.25 | -0.50 | -0.75 | -1.00 | -1.24 | -1.48 | -1.72 | -1.96 | -2.20 |
|    | + 0   | 0.00  | 0.25  | 0.50  | 0.76  | 1.02  | 1.28  | 1.54  | 1.80  | 2.06  | 2.32  |
| °C | 50    | 2.58  | 2.85  | 3.11  | 3.38  | 3.65  | 3.92  | 4.19  | 4.46  | 4.73  | 5.00  |
| ↓  | 100   | 5.27  | 5.54  | 5.81  | 6.08  | 6.36  | 6.63  | 6.90  | 7.18  | 7.45  | 7.73  |
|    | 150   | 8.00  | 8.28  | 8.56  | 8.84  | 9.11  | 9.39  | 9.67  | 9.95  | 10.22 | 10.50 |
|    | 200   | 10.78 | 11.06 | 11.34 | 11.62 | 11.89 | 12.17 | 12.45 | 12.73 | 13.01 | 13.28 |
|    | 250   | 13.56 | 13.84 | 14.12 | 14.39 | 14.67 | 14.94 | 15.22 | 15.50 | 15.77 | 16.05 |
|    | 300   | 16.33 | 16.60 | 16.88 | 17.15 | 17.43 | 17.71 | 17.98 | 18.26 | 18.54 | 18.81 |
|    | 350   | 19.09 | 19.37 | 19.64 | 19.92 | 20.20 | 20.47 | 20.75 | 21.02 | 21.30 | 21.57 |
|    | 400   | 21.85 | 22.13 | 22.40 | 22.68 | 22.95 | 23.23 | 23.50 | 23.78 | 24.06 | 24.33 |
|    | 450   | 24.61 | 24.88 | 25.16 | 25.44 | 25.72 | 25.99 | 26.27 | 26.55 | 26.83 | 27.11 |
|    | 500   | 27.39 | 27.67 | 27.95 | 28.23 | 28.52 | 28.80 | 29.08 | 29.37 | 29.65 | 29.94 |
|    | 550   | 30.22 | 30.51 | 30.80 | 31.08 | 31.37 | 31.66 | 31.95 | 32.24 | 32.53 | 32.62 |
|    | 600   | 33.11 | 33.41 | 33.70 | 33.99 | 34.29 | 34.58 | 34.88 | 35.18 | 35.48 | 35.78 |
|    | 650   | 36.08 | 36.38 | 36.69 | 36.99 | 37.30 | 37.60 | 37.91 | 38.22 | 38.53 | 38.84 |
|    | 700   | 39.15 | 39.47 | 39.78 | 40.10 | 40.41 | 40.73 | 41.05 | 41.36 | 41.68 | 42.00 |

Voltages are listed in millivolts, or mV.

Table 7-5

Voltage-versus-temperature for Type J: Iron-Constantan thermocouples.

The solution to this problem is to measure the actual reference temperature with an RTD or thermistor as shown in Figure 7-14. The voltage corresponding to the reference temperature is looked-up in the thermocouple table and added to the voltage seen by the signal conditioning circuit. This sum is called the **corrected voltage** and is then used to determine the thermocouple temperature.

**Example 7-4:**

Suppose that a thermistor is used to measure the reference temperature for a type J thermocouple. The thermistor indicates that the reference temperature is room temperature, or 25°C. What is the thermocouple temperature if the voltage measured by the signal conditioning circuit in Figure 7-14 is 14.22 mV?

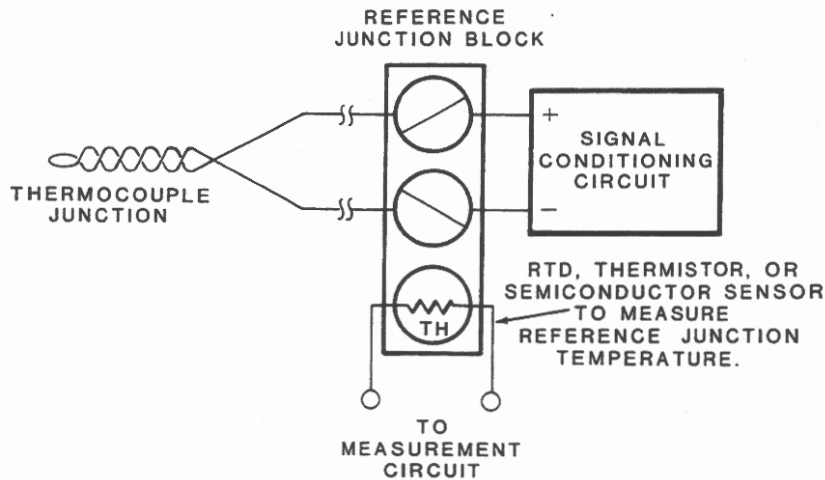


Figure 7-14

Measuring the reference junction temperature, using an RTD, thermistor, or semiconductor temperature sensing device.

**Solution:**

Since the thermistor indicates a reference temperature of 25°C, the reference voltage from Table 7-5 is 1.28 mV. Adding this to the measured voltage of 14.22 mV, gives a corrected voltage of:

$$\begin{aligned}
 \text{Corrected Voltage} &= \text{Measured Voltage} + \text{Reference Voltage} \\
 &= 14.22 \text{ mV} + 1.28 \text{ mV} \\
 &= 15.50 \text{ mV}
 \end{aligned}$$

Now, using Table 7-5 again, you find that the corrected voltage of 15.50 mV corresponds to a temperature of about 285°C. This is the actual thermocouple temperature.

The above correction procedure is called **software compensation**, since a microprocessor is programmed to perform the operation using memory look-up tables like the one in Table 7-5. Software compensation is the most versatile technique used with thermocouples. But, as you can see, it involves the measuring of the reference junction temperature and the thermocouple voltage, to get the thermocouple temperature.

As mentioned earlier, electrical noise is a major problem when thermocouples are used in the industrial environment. The thermocouple acts like an antenna that picks up industrial noise generated by large electrical machines, as well as radio frequency (rf) noise, TV, and other communications transmissions. The noise induced into the thermocouple may be greater than the actual thermocouple output voltage. The most common method used to reduce noise is to enclose the thermocouple in a grounded stainless steel sheath as shown in Figure 7-15, and use an op amp instrumentation amplifier to amplify the thermocouple output. The grounded sheath shields the circuit from rf noise, and the instrumentation amplifier cancels out any remaining noise induced on the thermocouple. An instrumentation amplifier is a very versatile signal conditioning circuit.

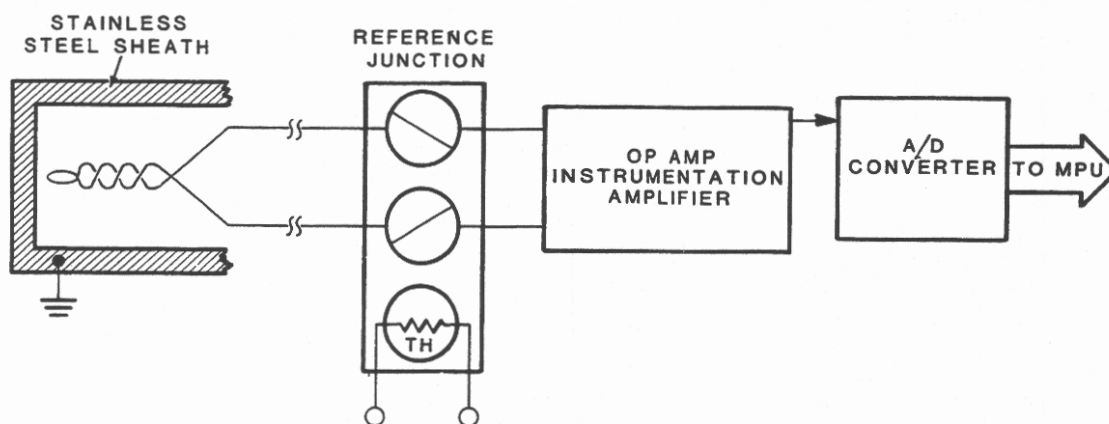


Figure 7-15

A grounded stainless steel sheath and instrumentation amplifier, are used with a thermocouple to reduce noise.

By now, you have probably questioned the need to use a thermocouple. Since another device is required anyway, to determine the reference junction temperature, why not use either a RTD, thermistor, or semiconductor junction temperature sensor instead? The answer to this logical question is that other temperature sensing devices are only accurate within a certain temperature range, usually less than 300°C. On the other hand, thermocouples can be used to measure these temperatures, as well as extreme temperatures commonly found in many industrial processes. In addition, thermocouples are very small, rugged, economical, and respond in milliseconds rather than seconds to temperature changes.

## Semiconductor Junction Devices

Recently, semiconductor junction devices have become popular as temperature sensors. Standard diodes and transistors can be used, and special temperature sensing ICs are also available. These devices can be used in either of two ways — as thermoresistive or thermoelectric devices.

When you reverse-bias the PN junction of a diode or transistor, its reverse-bias resistance is a function of temperature. Also, like a thermistor, its reverse-bias resistance is inversely proportional to temperature. For example, if you connect an ohmmeter to a standard signal diode as shown in Figure 7-16, its PN junction is reverse-biased. If the ohmmeter is set to a high resistance range and the diode is held over a flame, you will observe that the reverse resistance of the diode is very high at room temperature, but drops very rapidly towards zero when exposed to the high temperature of the flame. You will even notice a change in resistance by grasping the diode between two fingers. (Try it!) When used in this way, the diode is acting like a thermistor.

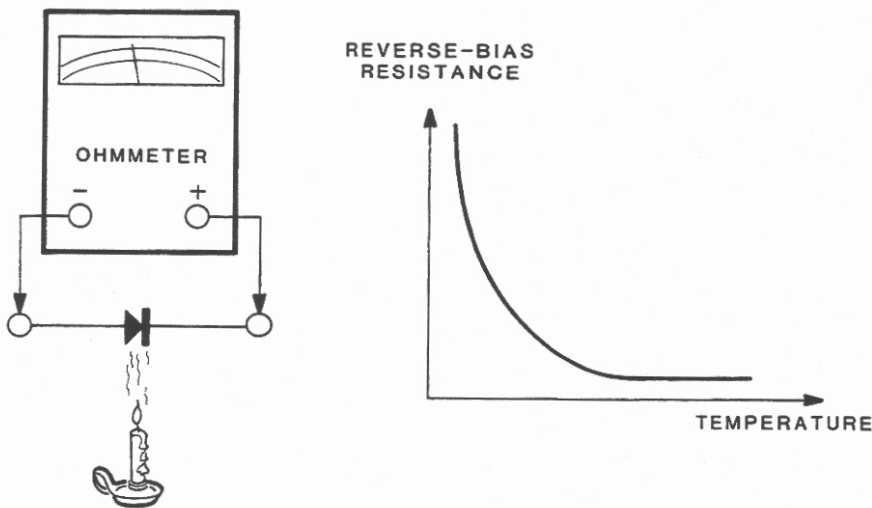


Figure 7-16

The reverse-bias resistance of a diode is inversely proportional to temperature.



Another way to use a diode as a temperature sensing device is to forward-bias it with a constant current source as shown in Figure 7-17. Notice how the forward voltage across the diode junction varies with the change in temperature. Depending on the type of diode used, the voltage variation will be between 1.5 mV and 2.5 mV for each  $1^{\circ}\text{C}$  temperature change. Moreover, the voltage variation will be linear from  $-50^{\circ}\text{C}$  to about  $125^{\circ}\text{C}$ . Of course, the voltage must be amplified before it can be measured by an A/D converter. An op amp instrumentation amplifier circuit can be used for this purpose. Many commercial semiconductor temperature sensing devices use this principle of operation.

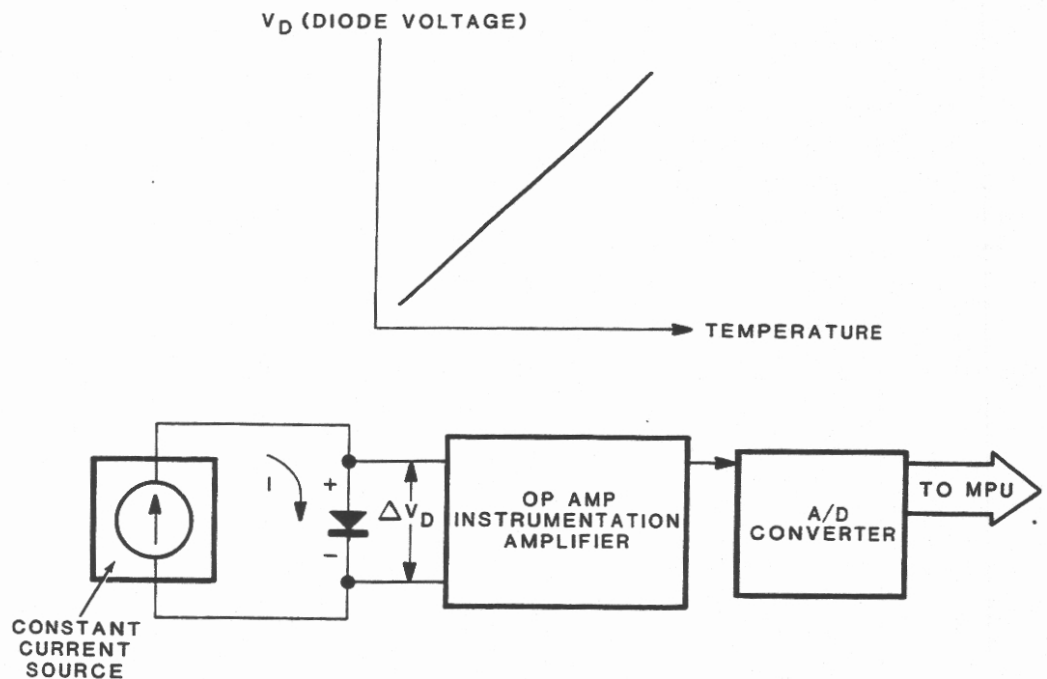


Figure 7-17

When a diode is forward-biased, the voltage drop across the diode will vary directly and linearly with temperature.

You can also use a general purpose transistor for temperature sensing as illustrated in Figure 7-18. This circuit operates on the principle that the current gain, or beta ( $\beta$ ), of a transistor changes with changes in temperature. As the beta of the transistor changes with temperature, the output voltage of the circuit in Figure 7-18 also changes. Moreover, you will find that the voltage changes are relatively linear with respect to temperature. The transistor's physical location can be fixed or mounted in a probe.

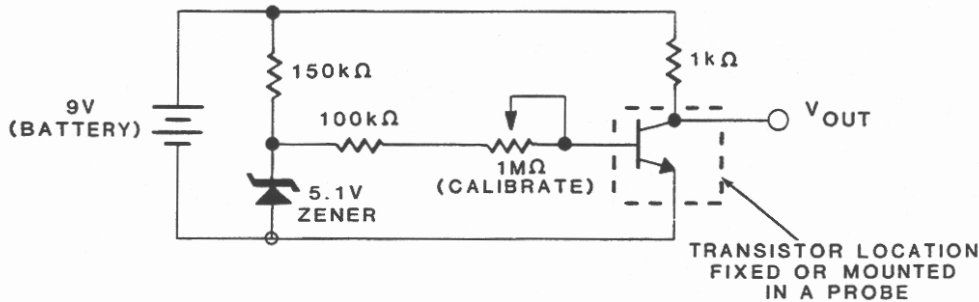


Figure 7-18

A standard general purpose transistor can be used as a temperature sensor.

The circuit output voltage,  $V_{out}$ , is proportional to temperature, since the  $\beta$  of the transistor changes with temperature.



## Thermoswitches

Earlier, you found that different materials have different temperature coefficients of resistance. In the same way, different materials have different coefficients of linear expansion. In other words, the length of a material changes with temperature. The relationship between length and temperature of a given material is expressed as its **temperature coefficient of linear expansion**, or  $\gamma$  (Greek, gamma). The temperature coefficient of linear expansion of a material is defined as the percentage of change in its length with changes in temperature. If you know the length of a conductor at a given temperature, you can find its length at another temperature using its linear expansion coefficient as follows:

$$L_{\text{NEW}} = L_{\text{OLD}} [1 + \gamma (T_{\text{NEW}} - T_{\text{OLD}})]$$

Where:  $\gamma$  = the temperature coefficient of linear expansion of the material in  $\%/^{\circ}\text{C}$

$L_{\text{OLD}}$  = the length of the material at temperature  $T_{\text{OLD}}$

$L_{\text{NEW}}$  = the length of the material at temperature  $T_{\text{NEW}}$

The temperature coefficients of linear expansion for several common materials are listed in Table 7-6.

| MATERIAL | $\gamma$<br>( $\%/^{\circ}\text{C}$ ) |
|----------|---------------------------------------|
| Copper   | $16.6 \times 10^{-4}$                 |
| Aluminum | $25 \times 10^{-4}$                   |
| Steel    | $6.7 \times 10^{-4}$                  |

Table 7-6

Temperature coefficient of linear expansion for several common materials used in bimetallic strips.

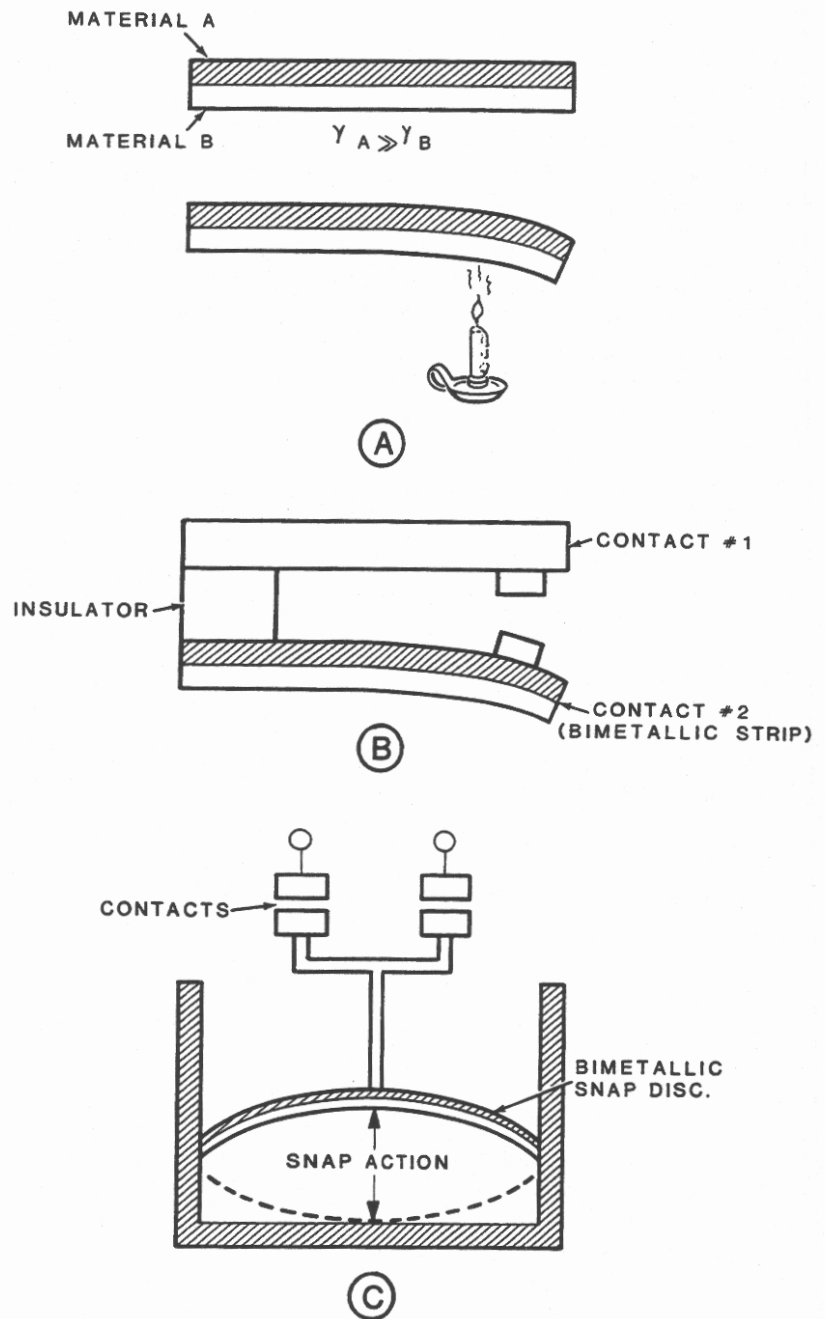


Figure 7-20  
A. Bimetallic-action,  
B. contact strip,  
C. and snap disc.

## CONSTRUCTION, OPERATION, AND CHARACTERISTICS

When two materials with different linear expansion temperature coefficients are bonded together, the different expansion rates cause the **thermoswitch** to bend when heated as illustrated in Figure 7-20A. The bending action is used to make or break a contact at some predetermined temperature. This principle is commonly used in thermostats and is illustrated in Figure 7-20B.

Thermoswitches can take the form of bimetallic reed switches as shown in Figure 7-20B or bimetallic discs as shown in Figure 7-20C. Bimetallic discs are used to provide a snap action to open or close a set of contacts. Thermoswitch devices can be designed to operate from sub-zero temperatures to several hundred degrees above 0°C. This type of sensor does not lend itself to temperature measurement with a microprocessor. However, it can be used to indicate a given temperature by opening or closing a set of contacts. The opening or closing of the contacts could be used to interrupt the MPU to perform a control task.

Thermoswitch devices are easily interfaced to microprocessor interrupt inputs or input ports via pull-up/down resistor circuits.

## LIQUID EXPANSION THERMOSWITCHES

Liquid expansion thermoswitches are familiar to all of us, since the principle of liquid expansion with temperature is used in the everyday household thermometer. One of the most common liquids used in thermometers is mercury. Aside from its large and extremely linear expansion coefficient, mercury is also a conductor. Consequently, mercury is commonly used in temperature switches called **mercury switches**. A mercury switch is illustrated in Figure 7-21A. Here, the level of mercury in the column is used to make or break the circuit at a predetermined temperature. The temperature at which the circuit is closed is changed if the length of the mercury column or the spacing of the mercury contacts is changed.

Like other thermoswitch devices, liquid expansion sensors do not lend themselves to temperature measurement in a microprocessor system. However, they can be used to indicate a given temperature by making or breaking a circuit. Control to within .05°C is possible with commercial devices. For example, the pull-up resistor circuit shown in Figure 7-21B could be used to generate an interrupt directly to the MPU when the level of mercury in the switch column reaches the top contact at some predetermined temperature. This particular circuit generates an active low, or logic 0, interrupt signal to the MPU.

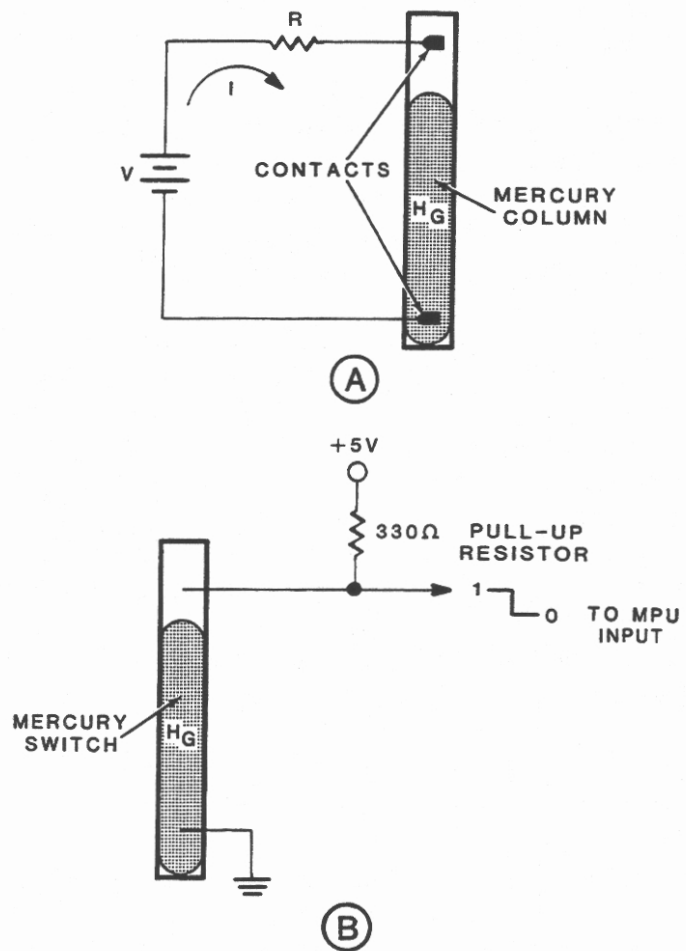


Figure 7-21

- A. A mercury column used to make or break a circuit.
- B. A mercury column generating a signal to a computer-based controller.

The features of the various temperature sensors discussed in this section are summarized by Table 7-7.


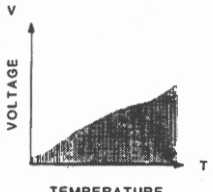

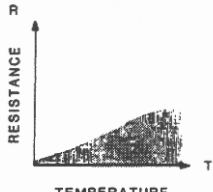

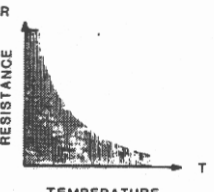

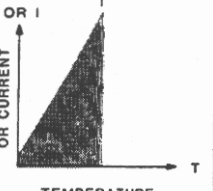
|               | THERMOCOUPLE<br><br>                                                          | RTD<br><br>                                                    | THERMISTOR<br><br>                                       | I.C. SENSOR<br><br>                                           |
|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ADVANTAGES    | <input type="checkbox"/> SELF-POWERED<br><input type="checkbox"/> SIMPLE<br><input type="checkbox"/> RUGGED<br><input type="checkbox"/> INEXPENSIVE<br><input type="checkbox"/> WIDE VARIETY<br><input type="checkbox"/> WIDE TEMPERATURE RANGE | <input type="checkbox"/> MOST STABLE<br><input type="checkbox"/> MOST ACCURATE<br><input type="checkbox"/> MORE LINEAR THAN THERMOCOUPLE                                                                                         | <input type="checkbox"/> HIGH OUTPUT<br><input type="checkbox"/> FAST<br><input type="checkbox"/> TWO-WIRE OHMS MEASUREMENT                                                                                                | <input type="checkbox"/> MOST LINEAR<br><input type="checkbox"/> HIGHEST OUTPUT<br><input type="checkbox"/> INEXPENSIVE                                                                                                           |
| DISADVANTAGES | <input type="checkbox"/> NON-LINEAR<br><input type="checkbox"/> LOW VOLTAGE<br><input type="checkbox"/> REFERENCE REQUIRED<br><input type="checkbox"/> LEAST STABLE<br><input type="checkbox"/> LEAST SENSITIVE                                 | <input type="checkbox"/> EXPENSIVE<br><input type="checkbox"/> CURRENT SOURCE REQUIRED<br><input type="checkbox"/> SMALL $\Delta R$<br><input type="checkbox"/> LOW ABSOLUTE RESISTANCE<br><input type="checkbox"/> SELF-HEATING | <input type="checkbox"/> NON-LINEAR<br><input type="checkbox"/> LIMITED TEMPERATURE RANGE<br><input type="checkbox"/> FRAGILE<br><input type="checkbox"/> CURRENT SOURCE REQUIRED<br><input type="checkbox"/> SELF-HEATING | <input type="checkbox"/> $T < 200^{\circ}\text{C}$<br><input type="checkbox"/> POWER SUPPLY REQUIRED<br><input type="checkbox"/> SLOW<br><input type="checkbox"/> SELF-HEATING<br><input type="checkbox"/> LIMITED CONFIGURATIONS |

Table 7-7  
Summary of Common Temperature Sensors.



## Self-Test Review

1. List the four major classifications of temperature sensing devices:

\_\_\_\_\_ .

\_\_\_\_\_ .

\_\_\_\_\_ .

\_\_\_\_\_ .

2. The relationship between temperature and resistance in a conductor is called:

\_\_\_\_\_ .

3. What is the approximate resistance of a nickel RTD at 75°C?

\_\_\_\_\_ .

4. Suppose a nickel RTD is placed in a 12 V DC Wheatstone bridge with a calibration resistor and two fixed resistors of 10 k $\Omega$  each. What is the bridge output at room temperature if the bridge is balanced at 0°C?

\_\_\_\_\_ .

5. As temperature increases, the resistance of a thermistor:

\_\_\_\_\_ .

6. How can self-heating of an RTD or thermistor affect temperature measurement?

\_\_\_\_\_ .

7. A thermocouple is a \_\_\_\_\_ sensing device.

8. Suppose you are using a type J thermocouple to measure the temperature of an industrial process. Assuming the reference junction is at room temperature, you measure a voltage of 26.11 mV. What is the temperature of the process?

\_\_\_\_\_ .

9. When programmed into a computer, the procedure used to find the temperature in question 8 is called:

\_\_\_\_\_ .

10. Explain how an ordinary signal diode can be used as a temperature sensing device.

11. List three types of thermostats.

\_\_\_\_\_ .

\_\_\_\_\_ .

\_\_\_\_\_ .

## Answers

1. A. Thermoresistive.  
B. Thermoelectric.  
C. Semiconductor.  
D. Thermoswitch.
2. Temperature coefficient of resistance, or  $\alpha$ , of the conductor.
3. From Table 2-1, you find that the temperature coefficient of nickel is .0067. From Table 2-2, you find that the resistance of a nickel RTD at 0°C is 120 ohms. Thus, its resistance at 75°C is:

$$\begin{aligned}
 R_{\text{NEW}} &= R_{\text{OLD}} [1 + \alpha (T_{\text{NEW}} - T_{\text{OLD}})] \\
 &= 120 \, \Omega [1 + .0067 (75^\circ\text{C} - 0^\circ\text{C})] \\
 &= 120 \, \Omega [1 + .5] \\
 &= 180 \, \Omega
 \end{aligned}$$

4. The calibration resistance,  $R_{\text{cal}}$ , must be 120  $\Omega$  to balance the bridge at 0°C (See Table 7-2). Also, from Table 7-2, the RTD has a resistance of 140  $\Omega$  at room temperature. Therefore, the bridge output must be:

$$\begin{aligned}
 V_{\text{out}} &= \left[ \frac{R_2}{R_1 + R_2} - \frac{R_{\text{cal}}}{R_{\text{RTD}} + R_{\text{cal}}} \right] \times V_{\text{in}} \\
 &= \left[ \frac{10 \, \text{k}\Omega}{10 \, \text{k}\Omega + 10 \, \text{k}\Omega} - \frac{120 \, \Omega}{140 \, \Omega + 120 \, \Omega} \right] \times 12 \, \text{V} \\
 &= .48 \, \text{V}
 \end{aligned}$$

5. As temperature increases, the resistance of a thermistor decreases.
6. Self-heating can change the resistance of an RTD or thermistor, resulting in an erroneous temperature measurement.
7. A thermocouple is a thermoelectric temperature sensing device.

8. Since the reference temperature is room temperature, or 25°C, the reference voltage from Table 7-5 is 1.28 mV. The measured voltage is 26.11 mV. Adding the measured and reference voltages gives us the corrected voltage, or:

$$\begin{aligned}\text{Corrected Voltage} &= \text{Measure Voltage} + \text{Reference Voltage} \\ &= 26.11 \text{ mV} + 1.28 \text{ mV} \\ &= 27.39 \text{ mV}\end{aligned}$$

From Table 7-5 you find that the temperature of the Process is 500°C.

9. Software Compensation.
10. When reverse-biased, the junction resistance of a diode is inversely proportional to temperature. When forward-biased, the voltage across a diode junction is directly proportional to temperature.
11. Thermoswitches include bimetallic reed switches, bimetallic discs, and liquid expansion mercury switches.

## OPTICAL SENSING

Optical sensors are generally used to perform one of two functions: detect the presence (absence) of light, or measure the intensity of light. Light detection systems employ optical sensors to measure industrial process variables such as position, proximity, and motion. Light measurement systems, such as TV cameras, employ optical sensors to convert levels of light intensity into electrical signals that can be processed by other electronic circuits.

In this section, you will learn about the most common optical sensing devices used in industrial applications. They are: **photoconductive devices**, **photodiodes**, **phototransistors**, and **integrated optical sensing devices**.

### Photoconductive Devices

A **photoconductive device** is exactly what its name implies. In operation, its conduction depends on the amount of light striking the device. In other words, its operation is basically that of a light-sensitive resistor whose internal resistance changes in response to changes in light intensity. As a result, it is often referred to as a **photoresistive cell** or **light dependent resistor (LDR)**.

### CONSTRUCTION AND OPERATION

Photoconductive devices are manufactured by depositing a light-sensitive material onto a ceramic base as shown in the cross-section diagram in Figure 7-22. Typical light-sensitive materials include: cadmium sulfide (CdS), cadmium selenide (CdSe), lead sulfide (PbS), and silicon (Si). A pair of electrodes are then formed on top and at each end of the light-sensitive material for external lead connections to the device. Observe that the two electrodes are separated by an S-shaped portion of exposed light-sensitive material. Finally, the *cell* is enclosed in a plastic, metal, or glass case, depending on the application environment the device is designed to withstand. Most low cost LDRs are enclosed in a plastic case with a transparent window, to allow light to strike the light-sensitive material. (Notice in Figure 7-22, that the LDR symbol consists of a resistor symbol inside a circle. Two arrows are used to show that the device is light-sensitive.)

The operation of a photoconductive device is straightforward. As light strikes the photosensitive material, the light energy causes electrons to be released from their valence orbits. These free electrons lower the resistance of the material and allow it to be a better conductor of current. The higher the light intensity, the lower the device resistance and the higher its conductivity.

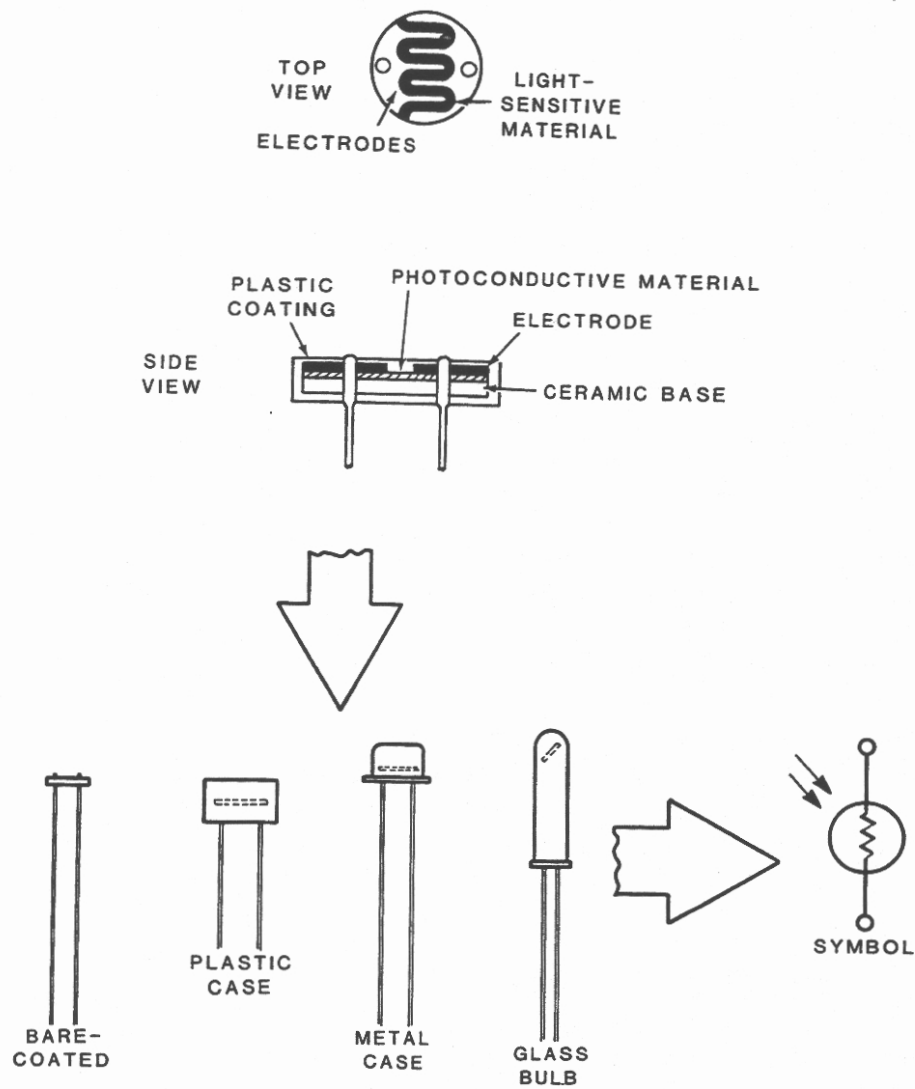


Figure 7-22  
Construction of a typical photoconductive cell, or LDR.

## CHARACTERISTICS

Photoconductive cells are more sensitive to light than any other type of light-sensitive device. The resistance of a typical cell might be as high as several hundred megohms in complete darkness, and as low as a few hundred ohms in normal room light. The sensitivity of one commercial device is illustrated by the graph in Figure 7-23. This graph shows the cell resistance versus illumination in lux, where lux is the unit of light incidence on the material. One lux equals one lumen per square meter, or  $1 \text{ lx} = 1 \text{ lm/m}^2$ .

Actually, the sensitivity of a given photoconductive device depends on the wavelength of light striking the device. Different light-sensitive materials exhibit different responses to different wavelengths of light. The graphs in Figure 7-24 show how the different light-sensitive materials respond to different types of light. The dashed line is visible light. Notice that the cadmium sulfide response curve almost matches that of visible light. For this reason, cadmium sulfide (CdS) is frequently used when the response must simulate that of the human eye. On the other hand, cadmium selenide (CdSe) is highly sensitive in the invisible infrared region.

Although photoconductive devices are very sensitive to changes in light, their biggest disadvantage is the fact that they respond slowly to light changes. In fact, they have the slowest response time of all light-sensitive devices. Also, they have a light-memory, or history effect. In other words, when the light level changes, the cell tends to "remember" the last illumination level rather than change rapidly to the new level.

Environmental conditions should always be considered when using any sensing device. Fortunately, the response of a photoconductive device is not influenced by temperatures between  $0^{\circ}\text{C}$  to  $50^{\circ}\text{C}$ . However, the life of the device is usually shortened when the light-sensitive material is exposed to long periods of high humidity or intense ultraviolet light.

Typical LDRs have maximum voltage ratings of 100, 200, or 300 volts DC. However, the maximum power consumption for these devices is relatively low. Maximum power ratings of 30 milliwatts to 300 milliwatts are common. The maximum power rating of a given device must not be exceeded, since permanent damage will usually result.

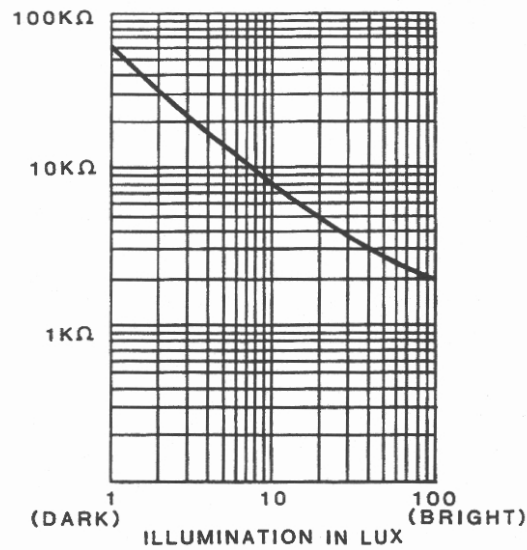


Figure 7-23

The sensitivity of a typical LDR is from several megohms in the dark to just a few ohms in bright light.

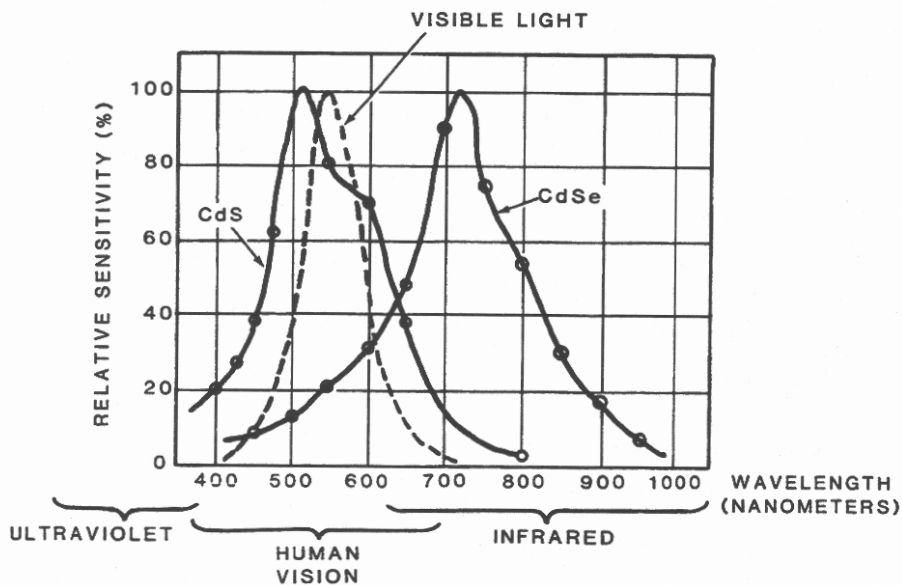


Figure 7-24

Different light sensitive materials exhibit different responses to light.



## APPLICATION CIRCUITS

Photoconductive devices are commonly used for any application that requires the measurement of light levels. Typical applications include: electronic camera shutters, outdoor day/night lighting, and robotic light level sensors, just to mention a few. In most cases, a photoconductive LDR device is placed in a voltage divider circuit as shown in Figure 7-25.

In Figure 7-25A, the voltage across the LDR changes in proportion to the light intensity. This causes a corresponding change in the voltage divider output,  $V_{out}$ . In Figure 7-25B, an LDR is being used as part of the resistive feedback network of a non-inverting operational amplifier circuit. The gain of the amplifier is controlled by the resistance of the LDR. As a result, the op amp output ( $V_{out}$ ) is a function of light intensity.

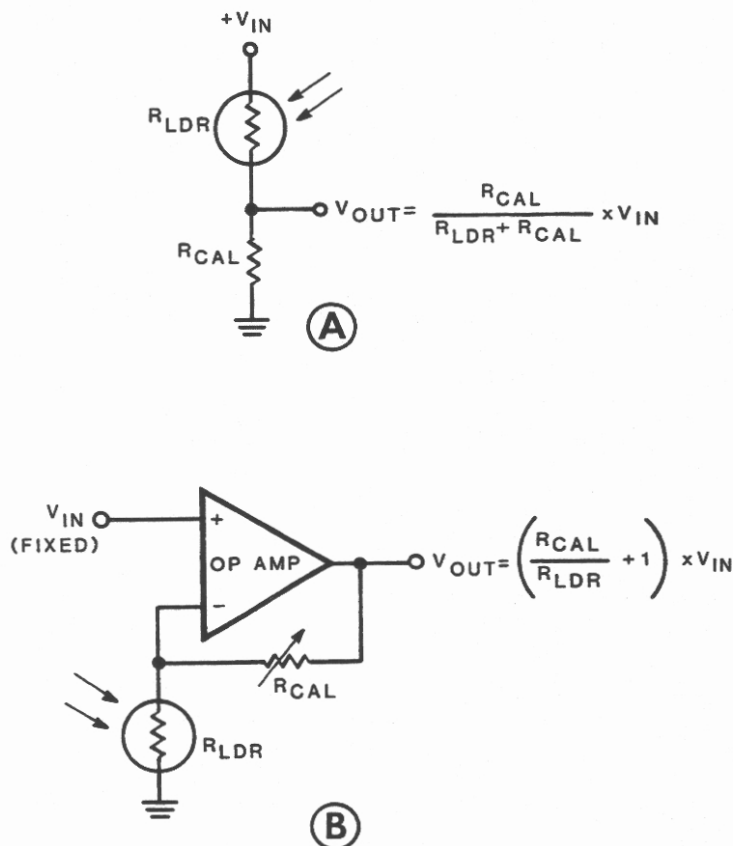


Figure 7-25

- A. An LDR as part of a voltage divider.  
 B. A non-inverting op amp circuit.

**Example 7-8:**

A given photoconductive cell has the following specifications:

$$\text{Dark Resistance} = 1 \text{ M}\Omega$$

$$\text{Light Resistance} = 10 \text{ k}\Omega$$

As the names imply, dark resistance is the resistance of the cell in complete darkness, while light resistance is its resistance in a bright light.

- A. Calculate the range of output voltages that you would expect from the voltage divider circuit in Figure 7-25A, when a  $1 \text{ M}\Omega$  fixed resistor is used in series with the photocell. Assume the input voltage is  $+12\text{V}$ .
- B. Calculate the range of output voltages that you would expect from the op amp circuit in Figure 7-25B, when  $R_{\text{cal}}$  is adjusted to  $100 \text{ k}\Omega$ . Assume the input voltage,  $V_{\text{in}}$ , is  $+1\text{V}$ .

**Solution:**

- A. In complete darkness the output voltage is:

$$\begin{aligned} V_{\text{out}} &= \frac{R_{\text{cal}}}{R_{\text{LDR}} + R_{\text{cal}}} \times V_{\text{in}} \\ &= \frac{1 \text{ M}\Omega}{1 \text{ M}\Omega + 1 \text{ M}\Omega} \times 12\text{V} \\ &= 6 \text{ V} \end{aligned}$$

In bright light the output voltage is:

$$\begin{aligned} V_{\text{out}} &= \frac{R_{\text{cal}}}{R_{\text{LDR}} + R_{\text{cal}}} \times V_{\text{in}} \\ &= \frac{1 \text{ M}\Omega}{10 \text{ k}\Omega + 1 \text{ M}\Omega} \times 12\text{V} \\ &= 11.88\text{V} \end{aligned}$$

Thus, the output range is from 6 V in complete darkness, to about 12 V in bright light.

B. In complete darkness the output voltage is:

$$\begin{aligned}V_{\text{out}} &= \left( \frac{R_{\text{cal}}}{R_{\text{LDR}}} + 1 \right) \times V_{\text{in}} \\&= \left( \frac{100 \text{ k}\Omega}{1 \text{ M}\Omega} + 1 \right) \times 1 \text{ V} \\&= 1.1 \text{ V}\end{aligned}$$

In bright light the output voltage is:

$$\begin{aligned}V_{\text{out}} &= \left( \frac{R_{\text{cal}}}{R_{\text{LDR}}} + 1 \right) \times V_{\text{in}} \\&= \left( \frac{100 \text{ k}\Omega}{10 \text{ k}\Omega} + 1 \right) \times 1 \text{ V} \\&= 11 \text{ V}\end{aligned}$$

Thus, the output voltage ranges from 1.1 V in complete darkness to 11 V in bright light.

## Photovoltaic Devices and Photodiodes

A photovoltaic device converts light energy directly into electrical energy. When exposed to light, these devices generate a small voltage across their terminals that increases as the light intensity increases. Photovoltaic devices are commonly called solar cells. Since they are often used to convert sunlight directly into electrical energy.

As you will soon discover, most photovoltaic devices are photodiodes. A **photodiode** is a semiconductor junction device that responds to light. As its name implies, it is constructed like a diode.

## CONSTRUCTION AND OPERATION

The basic construction of a PN photodiode is shown in Figure 7-26A. Observe that it consists of a P-type silicon region that has been grown into an N-type region to form a PN junction. A transparent non-reflective silicon dioxide window is placed over the P-type region, to allow light to enter the device. Electrical contact is made to the P-type layer through the silicon dioxide window via a metal contact. A metal backplate is attached to the N-type layer to allow electrical contact to this region. Notice from the associated symbol, that the P-type region is the anode, and the N-type region is the cathode of the photodiode.

Another type of construction is shown in Figure 7-26B. This type of photodiode is similar to the one in Figure 7-26A, with one major difference — the high resistance intrinsic, or I, layer between the P and N regions. Consequently, this type of construction is called a **PIN photodiode**.

PN and PIN photodiodes are often mounted on an insulative substrate, and sealed within a metal case as shown. A glass window is provided at the top of the case to allow light to enter and strike the diode.

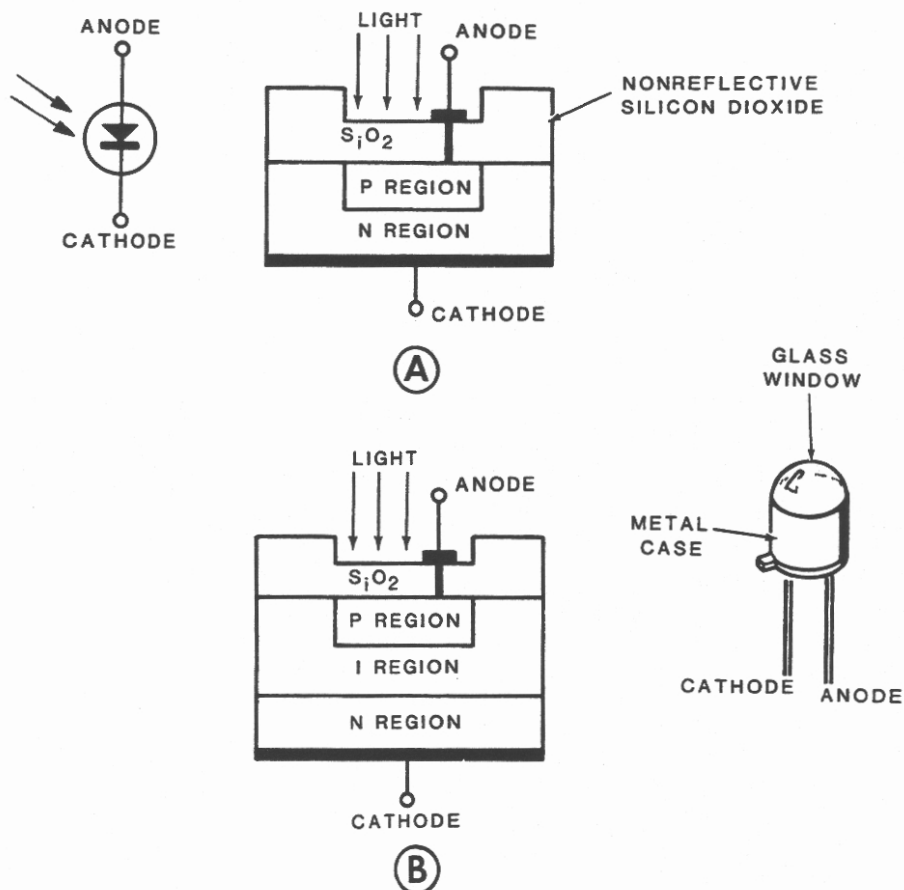


Figure 7-26

- A. The construction of a standard photodiode.
- B. The construction of a PIN photodiode.

Another type of photovoltaic device is the selenium photocell shown in Figure 7-27. Here, a junction is formed between two dissimilar materials: selenium and a metal base plate. A metal ring is attached to the selenium layer to act as a light window and allow electrical contact to the device. A lead is attached to the metal base plate to serve as the second electrical contact to the device. Although a selenium photocell is a photovoltaic device, it is not a photodiode since it does not contain a PN semiconductor junction.

Photodiodes can be operated in one of two modes: the **photovoltaic mode** or the **photoconductive mode**. In explaining the operation of photodiodes we will use the standard PN junction device, shown in Figure 7-26A, since it closely approximates the operation of other diodes you have studied previously.

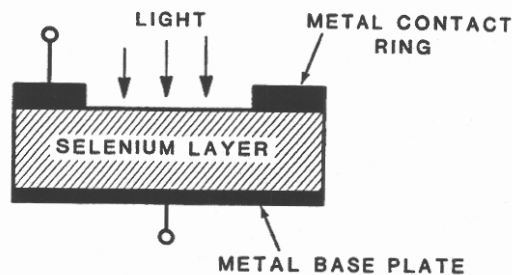


Figure 7-27

A selenium photovoltaic device.

## THE PHOTOVOLTAIC MODE

In its photovoltaic mode, the diode must convert light directly into voltage. In order to generate a voltage, the P-type layer of the photocell must be exposed to light. Light energy consists of many tiny particles called **photons**. When photons strike the P-type layer, they collide with atoms within the material and knock electrons out of their valence orbits. An electron knocked-out of its valence orbit becomes a free conduction band electron. This leaves the respective atom positively charged and a hole is created. The free electron and hole form an **electron-hole pair** which can drift through the semiconductor material.

Many of the electron-hole pairs are produced at the junction of the P and N layers. Free electrons on the N side of the junction are swept across to the P side, since unlike charges attract. This leaves positive ions on the N side of the junction, shown as a plus sign inside a circle in Figure 7-28A. Likewise, holes on the P side are swept across to the N side. This leaves negative ions on the P side of the junction, shown as a negative sign inside a circle.

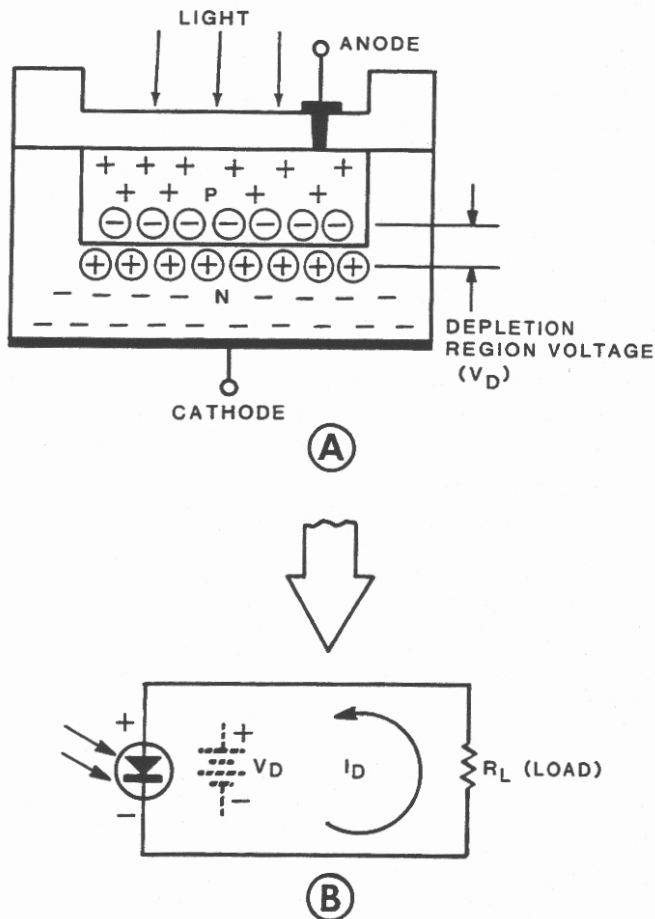


Figure 7-28

In its photovoltaic mode, a photodiode acts like a small battery.

The positive and negative ions form a **depletion region** around the junction. Since one side of the region is positively charged due to positive ions and the other side is negatively charged due to negative ions, there is a difference in potential, or voltage, developed across the region. This diode voltage,  $V_D$ , in Figure 7-28B is directly proportional to the size of the depletion region. As the depletion region grows, the diode voltage increases. What do you suppose causes the depletion region to grow? As you might have guessed, a higher light intensity striking the diode. So, as the light intensity changes, the size of the depletion region changes, which changes the voltage developed across the diode leads.

In fact, you can think of the diode as a small battery as shown in Figure 7-28B. Here, a load resistance ( $R_L$ ) has been connected to the diode so that a current ( $I_D$ ) flows in the circuit. The circuit current is proportional to the diode voltage ( $V_D$ ) by Ohm's law. Of course, the diode voltage is proportional to the light intensity due to the photoelectric effect.

## THE PHOTOCONDUCTIVE MODE

When operating in its photoconductive mode, the diode must be reverse-biased as shown in Figure 7-29. Reverse-bias causes a wide depletion region to form around the PN junction, since electrons in the N region are attracted to the positive terminal of the battery, and holes in the P region are attracted to the negative battery terminal. Because the region around the junction is "depleted" of electrons and holes, there is a high resistance across the junction. The larger the reverse-bias, the larger the junction resistance.

With a given reverse-bias potential and under dark conditions, the diode resistance is so high that little or no reverse current flows in the circuit. However, as light strikes the diode, free electrons and holes are created from the light energy. These electron-hole pairs lower the resistance of the junction. As a result, more *reverse current* can flow through the diode. Thus, the junction resistance is proportional to light intensity. As the light intensity changes, the junction resistance changes, which changes the reverse current flow in the diode circuit. You could say that a photodiode acts like an LDR when reverse-biased.

You should be aware that PIN photodiodes exhibit a higher reverse-bias resistance due to their high resistance intrinsic layer. This makes the PIN photodiode respond better to lower light frequencies (longer wavelengths). As a result, the PIN photodiode is more efficient over a wider range of the light spectrum. In addition, the PIN device has a lower internal capacitance which allows the device to respond faster to changes in light intensity.

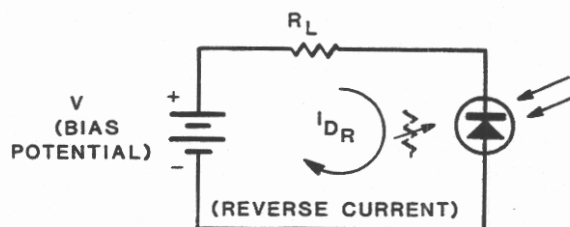


Figure 7-29

In its photoconductive mode, a photodiode acts like an LDR.

## CHARACTERISTICS

Photodiodes have an important advantage over the photoconductive devices described earlier. A photodiode can respond much faster to changes in light intensity. In fact, a photodiode operates faster than any other type of photosensitive device. This makes it useful in applications such as fiber optic data transmission and image analysis, where light fluctuates or changes intensity at a rapid rate.

## APPLICATION CIRCUITS

Photodiode applications are very common. They are used in photometers, spectrometers, video cassette recorders, medical instrumentation, computer communications, and solar cells in satellites. Two common application circuits are shown in Figure 7-30. In Figure 7-30A the photodiode is operating in its photovoltaic mode and connected to the input of a current-to-voltage op amp converter circuit. The current generated by the photodiode due to light intensity is amplified and converted to a voltage output,  $V_{out}$ . The output voltage is equal to the product of the diode current ( $I_D$ ) and the feedback resistor,  $R_F$ .

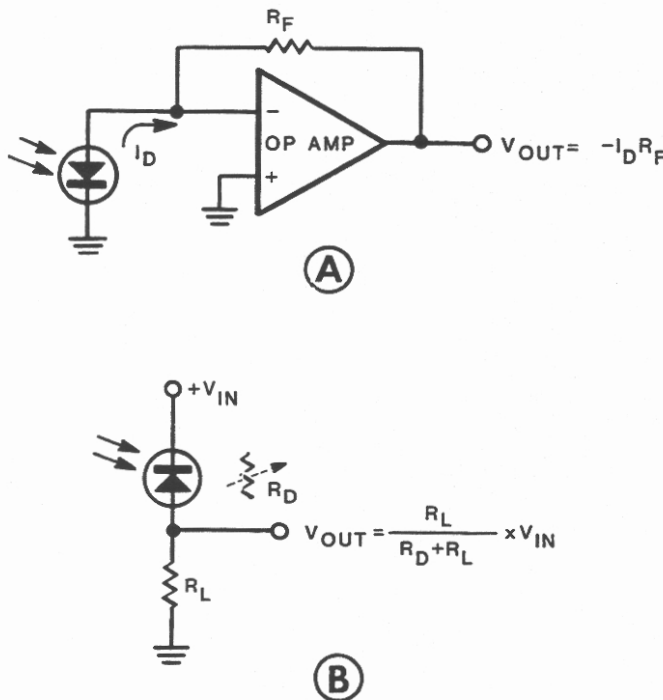


Figure 7-30

- A. A photodiode in its photovoltaic mode, drive an op amp.
- B. A photodiode in its photoconductive mode as part of a voltage divider circuit.



Observe that in Figure 7-30B, the diode is reverse-biased and, therefore, is operating in its photoconductive mode. This time, the photodiode is connected as part of a voltage divider. As the light intensity striking the diode changes, the diode resistance changes. This alters the output voltage,  $V_{out}$ , according to the voltage divider relationship shown in the figure.

Individual photodiodes are commonly used to detect the presence or absence of an object. Multiple photodiodes can be connected in a rectangular array fashion to form a **photodiode array** that can serve as an imaging device in a vision system. A vision system imaging device is an optical sensor that converts the light energy in a visual scene to electrical energy that can be analyzed by a computer.

#### Example 7-9:

Suppose a photodiode has the following specifications:

Dark current = 10 nA

Light current = 10  $\mu$ A

These current values are the amount of current the diode is expected to generate when connected to a load, and used in its photovoltaic mode.

Calculate the range of output voltages that you would expect to see generated from the op amp circuit in Figure 7-30A. Assume a feedback resistance,  $R_f$ , value of 1 M $\Omega$ .

#### Solution:

In complete darkness, the output voltage is:

$$\begin{aligned} V_{out} &= -I_D R_f \\ &= -(10\text{nA})(1\text{ M}\Omega) \\ &= -(10 \times 10^{-9}\text{ A})(1 \times 10^6\ \Omega) \\ &= -(10 \times 10^{-3}) \\ &= -.01\text{ V} \end{aligned}$$

In bright light the output voltage is:

$$\begin{aligned}V_{\text{out}} &= -I_D R_f \\&= -(10 \mu\text{A})(1 \text{ M}\Omega) \\&= -(10 \times 10^{-6} \text{ A})(1 \times 10^6 \Omega) \\&= -10 \text{ V}\end{aligned}$$

Consequently, the output voltage ranges from  $-0.01 \text{ V}$  in complete darkness to  $-10 \text{ V}$  in bright light.

## Phototransistors

The phototransistor is also a PN junction device. However, it has two junctions instead of one like the photodiode just described. The phototransistor is constructed in a manner similar to an ordinary transistor, but is used basically the same way as a photodiode in its photoconductive mode.

### CONSTRUCTION AND OPERATION

Bipolar phototransistors are constructed as shown in Figure 7-31. The process begins by taking an N-type silicon substrate, which ultimately serves as the transistor's collector, and diffusing a P-type region into the substrate to serve as the transistor's base. Then, an N-type region is diffused into the P-type base region to form the emitter.

As you can see, the phototransistor resembles a standard NPN bipolar transistor in appearance. The device is often packaged in a metal case with a glass window much like a photodiode. However, three leads are usually provided which connect to the collector, base, and emitter of the phototransistor. Some phototransistors omit the base lead, using only the emitter and collector leads.

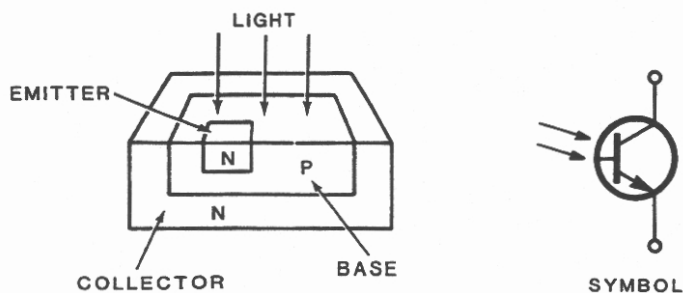


Figure 7-31

The construction and symbol of a typical phototransistor.

The operation of a phototransistor is easier to understand by considering the equivalent circuit shown in Figure 7-32. Notice that the circuit contains a photodiode connected across the base and collector junction of a conventional NPN bipolar transistor. If the equivalent circuit is biased by an external voltage as shown, electron current flows into the emitter lead and out of the collector lead. The amount of current flow is controlled by the transistor.

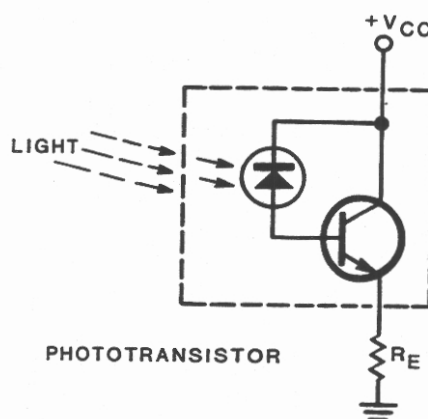


Figure 7-32

An equivalent circuit for understanding the operation of a phototransistor.

The transistor conducts more or less, depending on the conduction of the photodiode, which conducts more or less as the light striking it increases or decreases in intensity. Notice that the photodiode is reverse-biased and, therefore, is in its photoconductive mode. If light intensity increases, the diode conducts more current since its resistance decreases. This increases the emitter-to-base current (commonly called base current). From transistor theory, you know that a small increase in base current causes a large increase in the emitter-to-collector current flowing through the transistor due to the transistor's amplifying ability. As a result, any increase in light intensity causes a substantial increase in collector current. On the other hand, a decrease in light intensity causes a decrease in collector current.

Although the phototransistor may have a base lead as well as emitter and collector leads, the base lead is seldom used. When it is used, it is subjected to a bias voltage which will set the transistor's collector current to a nominal value under a given set of conditions. In other words, the base may be used to adjust the operating point of the phototransistor. However, in most applications you will only need to use the emitter and collector leads. Many times the base lead is simply "clipped-off" of the device.

## CHARACTERISTICS

An important difference between the phototransistor and photodiode is the amount of current that each device can deliver. The phototransistor can produce a much higher output current than a photodiode for a given light intensity because of its built-in amplifying ability. As a result, it is more sensitive to light variations and more useful for a wider variety of applications. Unfortunately, this higher sensitivity is offset by one important disadvantage. The phototransistor does not respond as quickly to changes in light intensity as the photodiode, and is not suitable for applications where an extremely fast response is required.

In addition to the bipolar junction-type phototransistor, field-effect phototransistors have also been developed. These devices are known as **photo-FETs**. Like other FET devices, photo-FETs have a very high input impedance. They also exhibit a much higher frequency response, meaning they can respond faster to changes in light intensity than bipolar devices. However, photoFETs have a very nonlinear response and their sensitivity drops rapidly as operating temperature increases.

## APPLICATION CIRCUITS

Like other types of photosensitive devices, the phototransistor is used in conjunction with a light source to perform many useful functions. It is widely used to measure position, linear velocity, angular velocity (rpm), and to count objects.

Two typical application circuits are shown in Figure 7-33. The phototransistor in Figure 7-33A is connected in series with a positive supply potential and an emitter resistance,  $R_E$ . The output,  $V_{out}$ , is taken across the emitter resistance,  $R_E$ . The voltage drop across this resistance is proportional to the light intensity striking the phototransistor, since the phototransistor controls the amount of current in the circuit. This type of circuit is commonly used to measure light intensity.

Another application circuit is shown in Figure 7-33B. Here, the phototransistor is connected to one input of an op amp comparator. The comparator generates pulses which are used to measure the rpm of a rotating object — a fan or propeller in this case. Each time the beam is broken by the rotating object, the comparator circuit generates an output pulse. These pulses can be counted by a digital circuit to determine the rpm of the object.

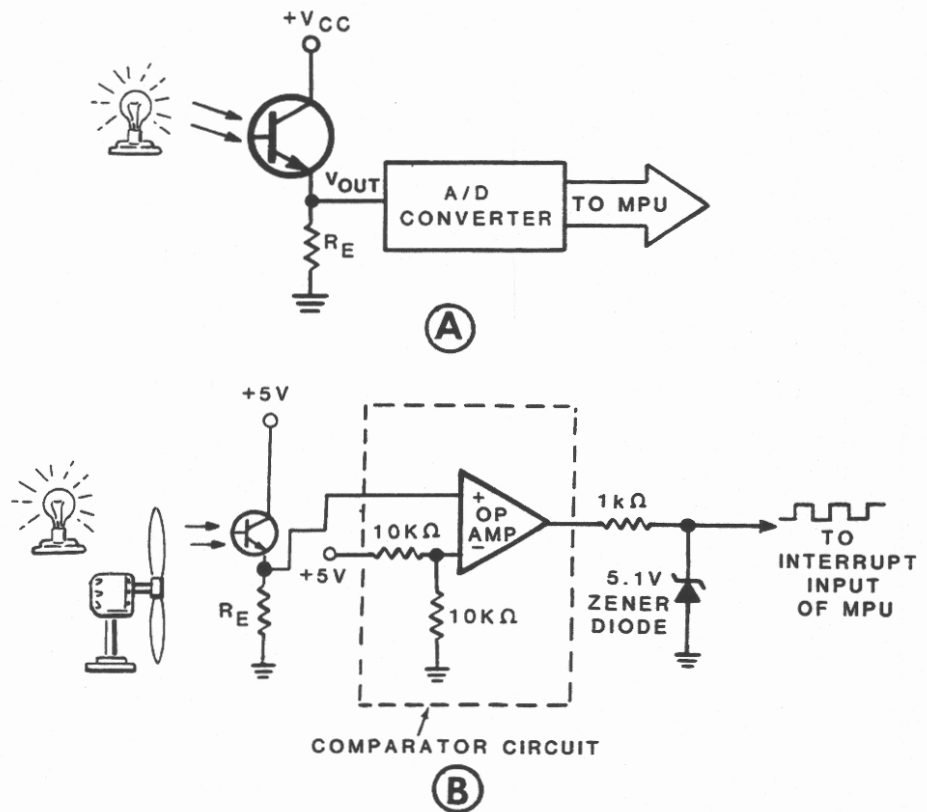


Figure 7-33

Two typical application circuits for a phototransistor.

A. A typical light sensor.

B. RPM measurement.

### Example 7-10:

Assume the phototransistor circuit in Figure 7-33A uses a +5 V supply and a 4.7 kΩ emitter resistance. Calculate the range of voltages that you would expect to see across the emitter resistance, from complete darkness to bright sunlight.

### Solution:

In complete darkness, the phototransistor is cut-off and acts like an open switch. Consequently, there is no voltage developed across the emitter resistance.

In bright sunlight, the phototransistor will be completely saturated, acting like a closed switch. Now, almost the entire +5 V supply is seen across the emitter resistance.

The ideal output voltage range is, therefore, 0 V to +5 V from darkness to bright sunlight. However, there will be a very small voltage developed in complete darkness due to current leakage within the transistor. Also, the voltage developed across the resistor in bright sunlight will be slightly less than 5 V, due to a small voltage drop within the transistor itself.

## Integrated Optical Sensing Devices

Integrated optical sensing devices combine a light source and light-sensing device in one molded integrated package. There are two types of integrated optical sensing devices that will be discussed here: the **optical interrupter module** and the **optical reflector module**.

### CONSTRUCTION AND OPERATION

Figure 7-34 shows both the optical interrupter and optical reflector modules. Look at the optical interrupter module in Figure 7-34A. As you can see from the electrical diagram, the module contains a light-emitting diode, or LED, and a phototransistor separated by a physical slot. The LED is typically an infrared LED, or IRED, and is mounted on one side of the slot. The phototransistor is mounted on the other side of the slot, directly opposite the IRED.

Observe how the optical reflector module in Figure 7-34B differs from the optical interrupter. Here, there is no slot and the IRED and phototransistor are mounted side-by-side within the module.

In operation, the IRED in both types of modules is constantly illuminated. In the case of the optical interrupter, a constant illumination falls on the phototransistor as long as the slot is unobstructed. However, if an object is placed within the slot to block the light beam, the phototransistor detects the decrease in light level. The resulting phototransistor output can then be used to stop or start machinery, trigger an alarm, supply count pulses, or perform other functions dictated by the device's application.

Optical reflector modules, as the name implies, operate from reflected light rather than from directly transmitted light. Notice from Figure 7-34B that the IRED and phototransistor are aimed so that their optical axes converge at a point just beyond the surface of the module. In order for the phototransistor to receive any light from the IRED, a reflective surface must be present near the focal point. Very little light will be detected by the phototransistor when a non-reflective dark area is present at the focal point. However, each time a reflective target reaches this point, a burst of light will be reflected back to the transistor, resulting in a change in the transistor's output.

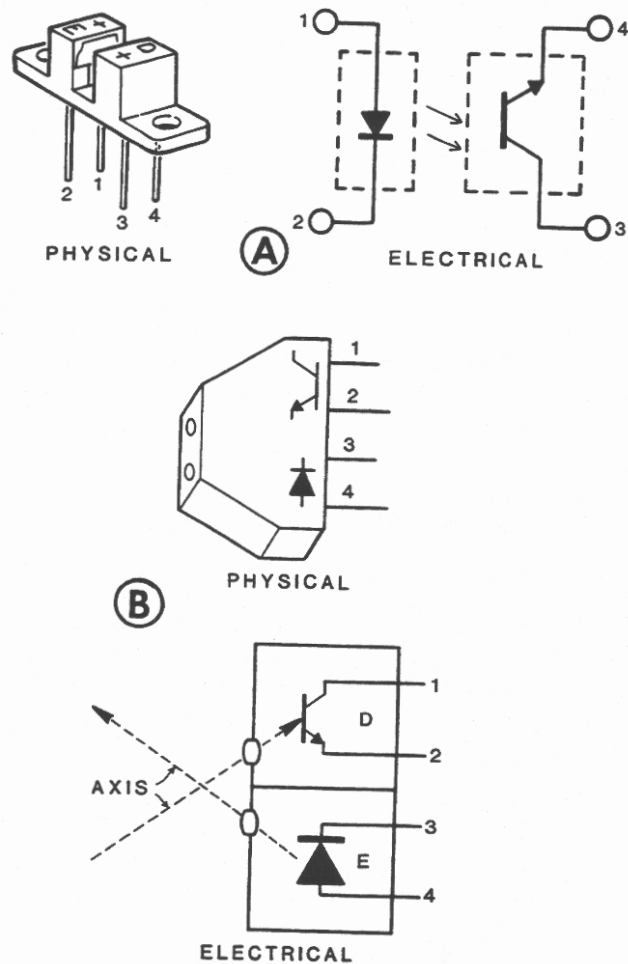


Figure 7-34

- A. An optical interrupter module.  
 B. An optical reflector module.

## CHARACTERISTICS

In most applications, discrete (separate) light source and light-sensing components can accomplish the same task as integrated interrupter and reflector modules. However, there are a number of advantages to combining the light source and light sensor into one package. First, you are assured that the light source and light sensor are matched within the light spectrum for maximum efficiency. Also, physical parameters such as alignment and source-to-sensor distance are optimized during the manufacturing of the device. Finally, another benefit is package standardization, which allows these devices to be economically used in mass production.

You should be aware that most optical interrupters have slot widths less than .375 inch, with .1 inch and .125 inch being very common. This limits the size of the object that can be detected by the device. Many times you must use optical encoder plates and wheels to detect larger objects. The optical reflector module does not have a slot to contend with, but its sensing distance is usually limited to less than .5 inch.

Most optical interrupter and reflector modules are designed to operate on voltage levels up to 20 V DC. This makes them compatible with voltage levels commonly found in digital control circuits.

## APPLICATION CIRCUITS

A typical application circuit that can be used for both an optical interrupter and reflector module is shown in Figure 7-35. Here, the IRED is forward-biased by the +5 V supply in series with a 100  $\Omega$  current limiting resistor. As a result, the IRED emits infrared light onto the base of the internal phototransistor. This causes the phototransistor to conduct, or saturate, and produce a +5 V potential at the top of the 4.7 k $\Omega$  resistor.

When the IRED light path is broken, the phototransistor does not conduct, or cuts-off, and a ground potential is seen at the top of the 4.7 k $\Omega$  resistor. You might think of the internal phototransistor as an optical switch which turns on and off with the presence or absence of light on its base. This particular circuit will generate a TTL pulse that goes from +5 V (logic 1) to 0 V (logic 0) each time the light path is broken. The logic transition can then be detected by a digital control circuit.

An op amp signal conditioning circuit has been added to the interrupter module output in Figure 7-35B. It turns out that the interrupter module output is rather noisy and full of glitches, especially when the light path is broken repeatedly at a high rate. Glitches in the output can cause false triggering of a subsequent digital circuit. The job of the op amp circuit is to clean-up the output so that it is acceptable to a digital control circuit. Of course, an optical reflector module could be substituted for the interrupter module in both the circuits in Figure 7-35.



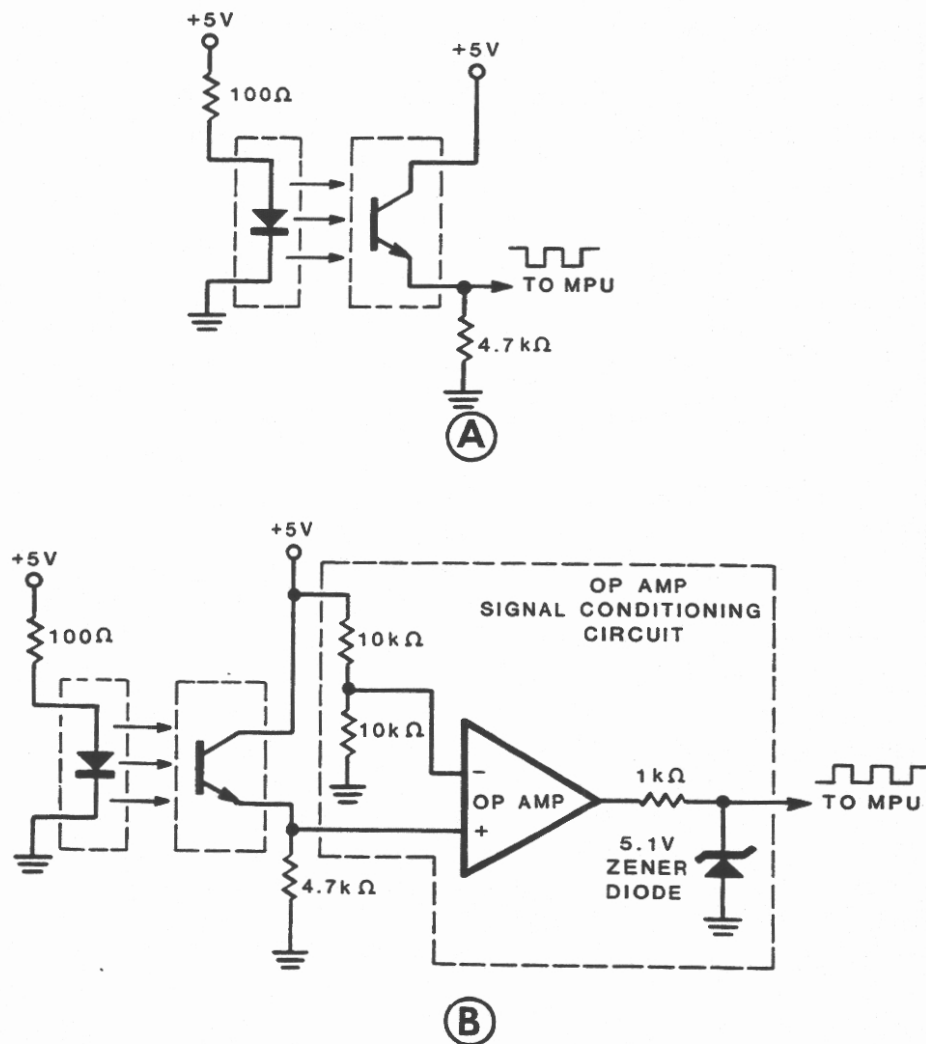


Figure 7-35

Common application circuits used with optical interrupter/reflector modules.

One example of how an interrupter module can be used is the optical tachometer shown in Figure 7-36. The objective, of course, is to measure the speed at which the shaft is rotating. An optical encoder wheel with a single slit is mounted to the shaft. An interrupter module is then positioned so that the edge of the encoder wheel is placed within the slot. As the wheel turns, the IRED light beam is blocked until the slit is within the slot. When this happens, a pulse appears on the output of the interrupter circuit. The number of pulses that occur in a given period of time is directly proportional to the speed of rotation. A microprocessor control circuit can count the pulses per unit time and be programmed to calculate the rpm.

In Figure 7-36B, this idea is extended to measure angular position by locating several slits at precise angular positions within the encoder wheel. Each slit causes the interrupter circuit to generate a pulse. By counting the pulses, a microprocessor can determine the angular position of the shaft.

In Figure 7-36C, linear position or travel is sensed by the interrupter module through the use of a linear position encoder strip. Again, several slits are located at precise linear positions on the encoder. As the encoder passes through the interrupter module, each slit causes a pulse to be generated. The pulses can then be counted to determine the linear position of a moving object. In addition, the amount of time between one pulse and the next, can be measured and used to determine the linear velocity of the object.

Another popular application of interrupter and reflector modules is position limit sensing. Mechanical limit switches are subject to malfunction due to friction, contact wear, and mechanical stress. By using optical interrupters and reflectors, these problems are avoided since no physical contact is required.

Interrupter modules are also useful for sensing the presence or absence of objects such as cards, tickets, tapes, or other opaque objects, capable of passing through the slot.

Many of the applications for interrupter modules can also be adapted to reflector modules. In addition, one novel application for a reflector module is the detection of slight movements and vibration. When used for this purpose, the device is placed so that a reflective object being tested is precisely at the module focal point. Since the light intensity at the phototransistor is a function of the distance to the object, very small changes in distance will result in large changes at the phototransistor output. We will explore this application more in the next unit when we learn about motion sensing.

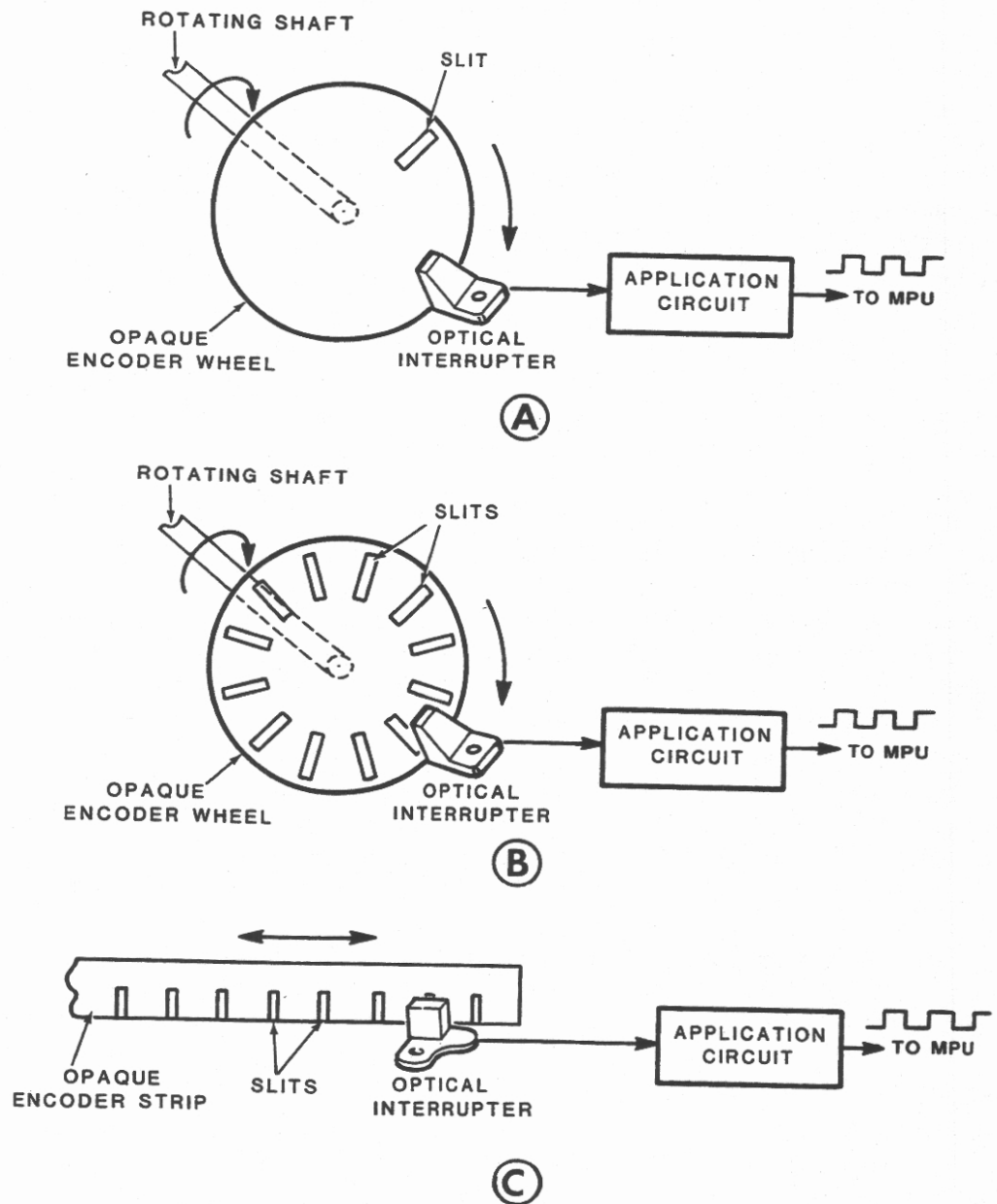


Figure 7-36

- A. An optical interrupter can be used to sense rpm.
- B. An optical interrupter sensing angular position and velocity.
- C. An optical interrupter sensing linear position and velocity.

A relatively recent application of an optical reflector module is its use as a medical heart rate monitor. For this application, an infrared reflector module is held against the skin. As the heart beats, the blood pulsations in the body cause the skin to reflect different levels of infrared light into the phototransistor. These levels of infrared light are then measured, and the output of the reflector module provides an electrical signal that follows the rhythm of the heartbeat. The output signal is finally amplified and displayed on an oscilloscope and/or recorded by a chart recorder.

A summary of the various optical devices discussed in this section is provided in Table 7-8.

Table 7-8

A comparison summary of the optical devices discussed in this section.

| <u>DEVICE</u>   | <u>ELECTRICAL VS. LIGHT CHARACTERISTICS</u>                                                                               | <u>APPLICATION CONSIDERATIONS</u>                                                                                                                                                                                                                                  |
|-----------------|---------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Photoconductive | Resistance changes inversely with light intensity.                                                                        | Very sensitive.<br>Wide resistance range.<br>Slow response.                                                                                                                                                                                                        |
| Photovoltaic    | Generates a voltage that increases as light intensity increases.                                                          | Fast response.<br>Used in solar cells.<br>Nonlinear.                                                                                                                                                                                                               |
| Photodiode      | Resistance changes inversely with light intensity when reversed biases.<br>Acts as photovoltaic cell when forward biased. | Fastest response.<br>Low output.<br>Can be operated as a photoconductive or photovoltaic device.<br>Available as PN or PIN diode.                                                                                                                                  |
| Phototransistor | Conduction/amplification level changes directly with light intensity.                                                     | Higher current output than photodiode.<br>Slower response than photodiode.<br>Very sensitive.                                                                                                                                                                      |
| Integrated      | Optocouplers and interrupter/reflector devices.                                                                           | Optocouplers provide electrical isolation of several kilovolts.<br>Optical interrupter modules are used to sense the presence or absence of objects.<br>Optical reflector modules are used to detect movement and only require access to one surface of an object. |

## Self-Test Review

12. An LDR is a \_\_\_\_\_ optical sensing device.
13. Suppose a given LDR has a dark resistance of  $1\text{ M}\Omega$  and a light resistance of  $1\text{ k}\Omega$ . What range of output voltages would you expect from a voltage divider circuit when a  $1\text{ M}\Omega$  fixed resistor is used in series with the LDR? Assume the supply voltage is  $+5\text{ V}$  and the output voltage is taken across the series resistance.
14. The two operating modes of a photodiode are the \_\_\_\_\_ and \_\_\_\_\_ modes.
15. When operating in the photoconductive mode, a PIN device must be \_\_\_\_\_ biased.
16. Suppose a photodiode has a dark current rating of  $5\text{ nA}$  and a light current rating of  $50\text{ }\mu\text{A}$ . What range of output voltages would you expect to see generated from the op amp circuit in Figure 7-30A? Assume the feedback resistance value is  $100\text{ k}\Omega$ .
17. What is the major operational difference between a photodiode and a phototransistor?
18. Suppose the emitter of a phototransistor is connected to ground and its collector is connected to  $+5\text{ V}$  through a  $10\text{ k}\Omega$  collector resistance as shown in Figure 7-37.
  - A. What is  $V_{\text{out}}$  in complete darkness?
  - B. What is  $V_{\text{out}}$  in bright sunlight?

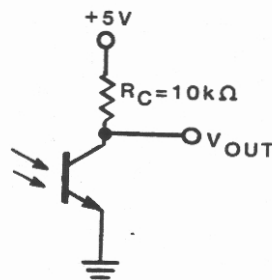


Figure 7-37

Circuit for Self-Test Review Question 18.

19. Two types of integrated optical sensing devices are:

\_\_\_\_\_.

\_\_\_\_\_.

20. An IRED is a \_\_\_\_\_.

21. Explain how an optical interrupter module can be used to measure angular position.

## Answers

12. Photoconductive Optical Sensing Device.

13. In complete darkness, the output voltage is:

$$\begin{aligned} V_{\text{out}} &= \frac{1 \text{ M}\Omega}{1 \text{ M}\Omega + 1 \text{ M}\Omega} \times 5 \text{ V} \\ &= 2.5 \text{ V} \end{aligned}$$

In bright light, the output voltage is:

$$\begin{aligned} V_{\text{out}} &= \frac{1 \text{ M}\Omega}{1 \text{ k}\Omega + 1 \text{ M}\Omega} \times 5 \text{ V} \\ &= 4.995 \text{ V} \end{aligned}$$

14. A. Photovoltaic.  
B. Photoconductive.
15. When operating in the photoconductive mode, a PIN device must be reverse-biased.
16. In complete darkness, the output voltage is:

$$\begin{aligned} V_{\text{out}} &= -I_d R_f \\ &= -(5 \text{ nA})(100 \text{ k}\Omega) \\ &= -.50 \text{ mV} \end{aligned}$$

In bright light, the output voltage is:

$$\begin{aligned} V_{\text{out}} &= -I_d R_f \\ &= -(50 \text{ }\mu\text{A})(100 \text{ k}\Omega) \\ &= -5 \text{ V} \end{aligned}$$

17. A Phototransistor can produce much higher output current levels than a photodiode for a given light intensity. However, a photodiode responds faster to changes in light intensity than a phototransistor.

18.
  - A. In complete darkness, the phototransistor is cut-off and does not conduct any current. Consequently, no voltage is dropped by the  $10\text{ k}\Omega$  collector resistance and the  $+5\text{ V}$  supply potential is seen at the output.
  - B. In bright sunlight, the phototransistor is saturated and conducts current. As a result, the  $10\text{ k}\Omega$  collector resistance drops all the  $+5\text{ V}$  supply potential and  $0\text{ V}$  is seen at the output.
19.
  - A. Optical Interrupter.
  - B. Optical Reflector Modules.
20. Infrared light-emitting diode.
21. See Figure 7-36B. Here, an optical encoder wheel is connected to a rotating shaft. The encoder wheel has several slits located at precise angular positions. When the wheel turns within the slot of the optical interrupter module, each slit causes the interrupter circuit to generate a pulse. The angular position is determined by keeping track (counting) of the number of pulses generated.



## UNIT SUMMARY

There are four major classifications of temperature sensors: thermoresistive, thermoelectric, semiconductor, and thermoswitches. Thermoresistive devices, such as RTDs and thermistors, change their resistance according to temperature changes. Thermocouples are thermoelectric devices that generate a small voltage which is proportional to temperature. Semiconductor junction diodes and transistors are used as either thermoresistive or thermoelectric temperature sensors, depending on how they are biased. Finally, thermoswitches made from dissimilar metals or mercury are used to make or break an electrical contact at some predetermined temperature.

Optical sensors are used extensively in industry to sense position and proximity. These light detection systems detect when a beam of light is broken by the presence of an object. This allows them to detect the presence of the object as well as its position using special optical encoder wheels and plates. Optical sensors are also employed in light measurement systems, such as vision systems, to measure light intensity.

The most common types of optical sensors used in industry are photoconductive devices, photodiodes, phototransistors, and integrated optical sensing devices. Photoconductive devices, such as LDRs, change their resistance in proportion to light intensity. Photodiodes can be operated in a photoconductive or photovoltaic mode. To operate in its photoconductive mode, the photodiode must be reverse-biased. In this mode, the diode junction resistance changes inversely with light intensity. In its photovoltaic mode, electron-hole pairs are created around the diode junction that produce a small voltage, which is directly proportional to light intensity. Phototransistors are used the same way as a photodiode in its photoconductive mode. However, because of its inherent amplifying ability, a phototransistor can produce a much higher current output than a photodiode for a given light intensity.

Finally, integrated optical sensors combine a light source and a light sensor in one molded integrated package. These devices include optical interrupter and reflector modules. In an optical interrupter module, the light source and light sensor are mounted directly opposite each other. In an optical reflector module, the light source and light sensor are mounted adjacent to each other.

*Unit 8*

**POSITION, PROXIMITY, AND  
FORCE SENSING**

## CONTENTS

|                                                 |      |
|-------------------------------------------------|------|
| Introduction .....                              | 8-3  |
| Unit Objectives .....                           | 8-4  |
| Position and Proximity Sensing .....            | 8-5  |
| Potentiometric Position Sensors .....           | 8-5  |
| Capacitive Position and Proximity Sensors ..... | 8-9  |
| Sensing Position .....                          | 8-11 |
| Sensing Proximity .....                         | 8-16 |
| Inductive Position and Proximity Sensors .....  | 8-17 |
| Sensing Position .....                          | 8-18 |
| The LVDT .....                                  | 8-20 |
| The RVDT .....                                  | 8-23 |
| Sensing Proximity .....                         | 8-24 |
| Magnetic Position and Proximity Sensors .....   | 8-25 |
| The Magnetic Reed Switch .....                  | 8-26 |
| Hall-Effect Devices .....                       | 8-28 |
| Sensing Position .....                          | 8-29 |
| Self-Test Review .....                          | 8-32 |
| Answers .....                                   | 8-33 |
| Force Sensing .....                             | 8-35 |
| Force .....                                     | 8-35 |
| Resistive Strain Gages .....                    | 8-37 |
| Construction and Operation .....                | 8-38 |
| Characteristics .....                           | 8-42 |
| Application Circuits .....                      | 8-42 |
| Semiconductor Strain Gages .....                | 8-48 |
| Construction and Operation .....                | 8-48 |
| Characteristics .....                           | 8-49 |
| Application Circuits .....                      | 8-49 |
| Piezoelectric Strain Gages .....                | 8-50 |
| Load Cells .....                                | 8-51 |
| Construction and Operation .....                | 8-52 |
| Characteristics .....                           | 8-53 |
| Application Circuits .....                      | 8-54 |
| Self-Test Review .....                          | 8-55 |
| Answers .....                                   | 8-57 |
| Unit Summary .....                              | 8-59 |

## Unit 8

# POSITION, PROXIMITY, AND FORCE SENSING

## INTRODUCTION

When a microprocessor is being used to control a process, it is often necessary to measure a mechanical activity such as position, proximity, and force. Although there are other mechanical phenomena that must be measured in many industrial processes, these three represent the majority of mechanical measurement requirements. (Consult the Heathkit/Zenith *Electronics For Automation* course for additional material on this subject).

In this unit, you will learn about electrical, optical, and magnetic sensors that are used to measure position and proximity. Position is the location of an object, while proximity is the nearness of an object. The role of the sensor is to generate an electrical signal that is proportional to the position or proximity being measured. This signal is usually an analog signal that must be converted to digital using an A/D converter for input to a microprocessor control system.

In the second section of this unit, you will learn how strain gages are used to measure force. A strain gage is a device whose resistance changes when it is subjected to an external force. The resulting resistance change is converted into a proportional analog voltage change by Ohm's Law. The voltage signal is then converted to digital by an A/D converter for input into a microprocessor control system.

## UNIT OBJECTIVES

1. Describe several common techniques used to measure the following mechanical phenomena:
  - Position
  - Motion
  - Force
2. Design magnetic detection circuits using magnetic reed switches and Hall effect devices.
3. Define stress and strain.
4. State the three general types of stress/strain situations.
5. Explain how resistive strain gages are used to measure force.
6. Define and calculate the gage factor for a resistive strain gage.

## POSITION AND PROXIMITY SENSING

Many industrial control processes require that the mechanical position or proximity (nearness) of an object be determined before a control decision can be made. This is especially true in robotics, where the robot controller must sense the position of its manipulator, with respect to its own reference coordinate system, and with respect to other objects within its work space.

We have grouped position and proximity sensing into this one section, because they are closely related and employ the same types of sensory devices and circuits. The discussion that follows is categorized by the types of devices that are used to sense position and proximity. While learning about these devices, you will see how they are applied to the position and proximity sensing tasks.

### Potentiometric Position Sensors

Potentiometers can be used to measure two types of position: **linear position** and **angular position**. The **rectilinear potentiometer** shown in Figure 8-1A is used to measure linear position, while the **angular potentiometer** in Figure 8-1B is used to measure angular position.

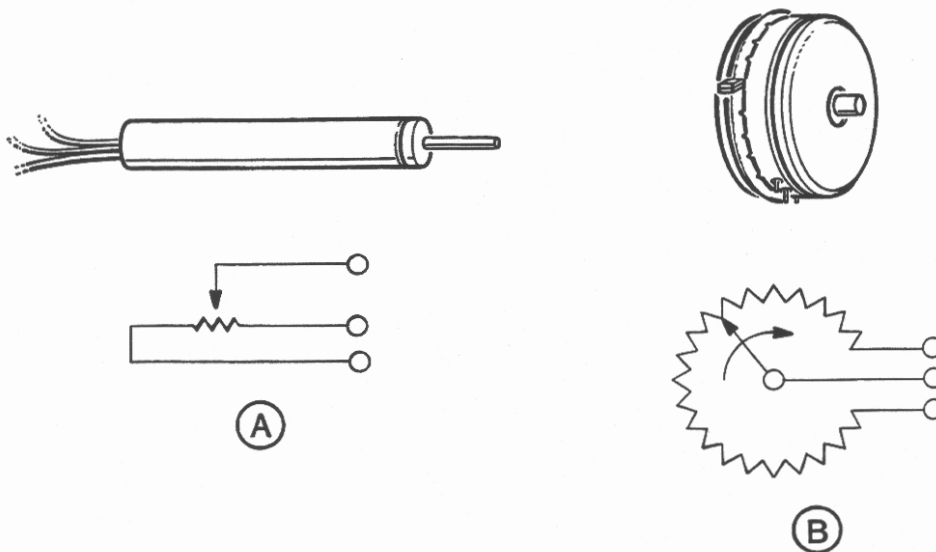
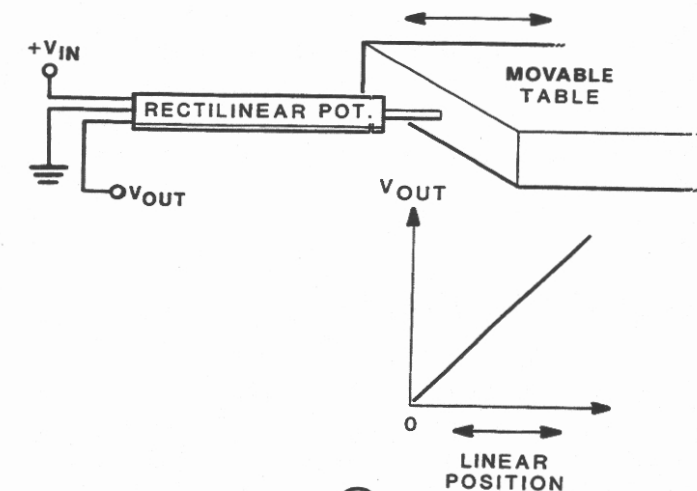


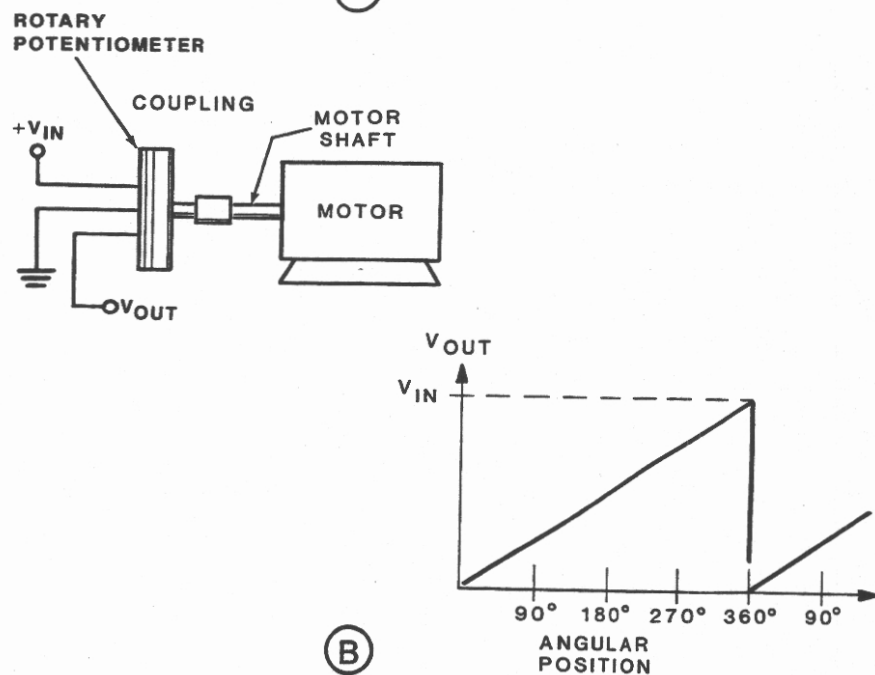
Figure 8-1

- A. A rectilinear potentiometer is used to measure linear position.
- B. A rotary potentiometer is used to measure angular position.

The basic idea is to attach the wiper of a potentiometer directly or indirectly to an object as shown in Figure 8-2. In Figure 8-2A, the wiper of a rectilinear potentiometer is attached to a moveable object and the output voltage,  $V_{out}$ , of the potentiometer is directly proportional to the linear position of the object. For instance, the wiper could be connected to a machine tool table to measure the linear "travel" of the table during a machining operation.



(A)



(B)

Figure 8-2

- A. Measuring linear position.  
B. Measuring angular position.

In Figure 8-2B, the wiper of a rotary potentiometer is attached to the rotating shaft of a motor. The wiper output voltage is directly proportional to the angular position of the motor. Notice how the angular potentiometer is designed so that its wiper voltage increases from 0 V to a maximum of  $V_{in}$  for every 360° rotation of the motor shaft.

In Figure 8-2 you see that an input voltage,  $V_{in}$ , must be applied to the potentiometers. This voltage must be extremely stable and well regulated to minimize measurement error. In addition, the resolution of the potentiometer affects the measurement accuracy. The resolution of a potentiometer is a function of the number of wire turns in the potentiometer and is calculated as follows:

$$\% \text{ Resolution} = 100/N$$

Where: N is the number of wire turns  
in the potentiometer.

**Example 8-1:**

- A. Calculate the resolution of a 1000 turn potentiometer.
- B. Given an input voltage of 10 V, determine the measurement accuracy, in volts, of the above potentiometer.

**Solution:**

- A. The resolution of a 1000 turn potentiometer is:

$$\begin{aligned}\% \text{ Resolution} &= 100/N \\ &= 100/1000 \text{ turns} \\ &= .1\%\end{aligned}$$

- B. With an input voltage of 10 V applied to the potentiometer, a .1% resolution translates to a measurement accuracy of:

$$(.1\%/100\%) \times 10 \text{ V} = .001 \times 10 \text{ V} = .01 \text{ V}$$

In other words, the smallest change in wiper voltage resulting from a change in its position is .01 V.



When using a potentiometer to measure position, you must be careful not to *load* the potentiometer output. This problem is illustrated in Figure 8-3. Here, a 10 k $\Omega$  potentiometer is supplied with an input voltage of 12 V. When the potentiometer is in the middle of its travel, you would expect to see a wiper voltage of 6 V, right? That depends, however, on the load circuit connected to the wiper.

In most applications, the load will be a signal conditioning circuit that conditions the wiper output prior to conversion by an A/D converter. If the signal conditioning circuit represents a load of 5 k $\Omega$ , the wiper voltage will only be 4 V, rather than 6 V, when it is in the middle of its travel. This is because the wiper voltage is developed across the *parallel equivalent* of the load and wiper resistances as shown in Figure 8-3A.

The solution to this *loading* problem is to use a signal conditioning circuit that has a very high input resistance. This can be accomplished by placing an operational amplifier circuit, called a **voltage follower**, between the potentiometer and the signal conditioning circuit as shown in Figure 8-3B. With this arrangement, the potentiometer "sees" a load of almost infinite resistance. As a result, the wiper voltage is not loaded.

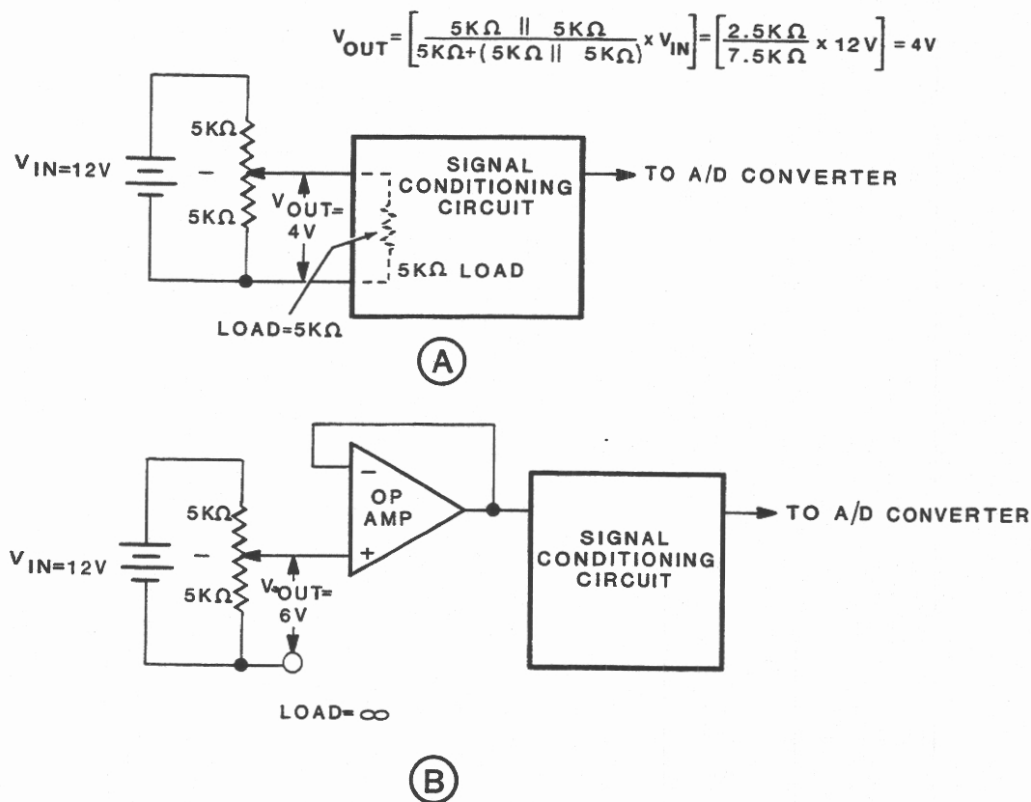


Figure 8-3

- A. The potentiometer loading problem.
- B. The solution to the loading problem.

## Capacitive Position and Proximity Sensors

Recall that a capacitor consists of two metal plates separated by an insulating material called the **dielectric**. The construction of a typical parallel plate capacitor is shown in Figure 8-4. A capacitor can be used for sensing position or proximity as a result of the following relationship:

$$C = \frac{.225 kA}{d}$$

where: C is capacitance in picofarads (pF).

k is the relative dielectric constant of the capacitor dielectric material (See Table 8-1).

A is the surface area of one plate in square inches (in<sup>2</sup>).

d is the distance between the plates in inches (in).

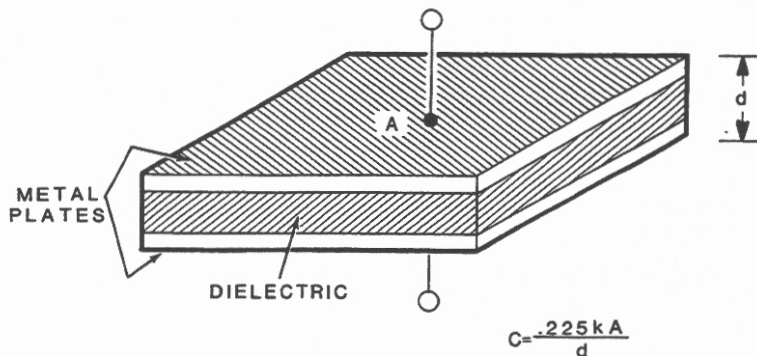


Figure 8-4  
A parallel plate capacitor.

Note that the previous relationship provides capacitance in picofarads ( $10^{-12}$  Farads). The relative dielectric constant,  $k$ , depends on the type of dielectric material employed in the capacitor. Table 8-1 lists relative dielectric constants for several common materials.

| Material       | Relative Dielectric Constant K |
|----------------|--------------------------------|
| Vacuum         | 1                              |
| Air            | 1                              |
| Mica           | 7                              |
| Paper          | 4                              |
| Glass          | 6                              |
| Porcelain      | 6                              |
| Aluminum oxide | 10                             |
| Tantalum oxide | 11                             |

Table 8-1

Relative dielectric constants for commonly used materials.

If any two of the previous three quantities ( $k$ ,  $A$ , or  $d$ ) are held constant and the third changes, the capacitance ( $C$ ) of the capacitor will change proportionally. This is the idea behind any capacitive sensor.

## SENSING POSITION

The diagram in Figure 8-5 shows one way that a capacitive sensor can be used to measure linear position. Here, a moving object such as a machine tool table, is attached to a moveable plate of a capacitor that has an air dielectric. The other capacitor plate is held stationary. As the object moves, the distance between the plates of the capacitor changes, resulting in a change in capacitance.

Now, recall that the resistance a capacitor offers to the flow of alternating current is called **capacitive reactance**, or  $X_C$ , and is defined as follows:

$$X_C = \frac{1}{2\pi fC}$$

Where:  $X_C$  is the capacitive reactance in ohms

$$\pi = 3.14$$

$f$  is the frequency of the applied AC voltage in Hertz (Hz).

$C$  is the capacitance in Farads (F).

So, if a constant frequency AC source is applied to a capacitor, the capacitive reactance of the capacitor changes as its capacitance changes. By Ohm's law this results in a proportional change in the voltage drop across the capacitor. From the formula for capacitance ( $C$ ) given earlier, you can determine that as the distance between the plates increases in Figure 8-5, the capacitance decreases. A decrease in capacitance causes the capacitive reactance to increase, resulting in an increased voltage drop across the capacitor due to Ohm's law.

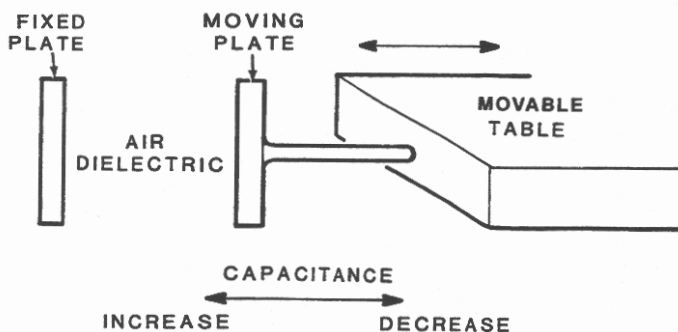


Figure 8-5

Measuring linear position by changing the distance between the plates of a capacitor.

**Example 8-2:**

Suppose a capacitive position sensor employs a moveable plate whose surface area is  $10 \text{ in}^2$ .

- A. Calculate its capacitance for an air dielectric when the moveable plate is positioned .1 in. from the stationary plate.
- B. Repeat the above calculation for a .5 in. position.
- C. Calculate the capacitive reactance that results in parts A and B, when the capacitor is driven by a 1 kHz AC source.

**Solution:**

- A. The relative dielectric constant for air is approximately 1 (See Table 8-1). Thus,

$$\begin{aligned} C &= \frac{.225 \text{ kA}}{d} \\ &= \frac{.225(1)(10 \text{ in}^2)}{.1 \text{ in.}} \\ &= 22.5 \text{ pF} \end{aligned}$$

- B. For a .5 in. position the capacitance is:

$$\begin{aligned} C &= \frac{.225 (1) (10 \text{ in}^2)}{.5 \text{ in.}} \\ &= 4.5 \text{ pF} \end{aligned}$$

- C. For the 22.5 pF capacitance:

$$\begin{aligned} X_C &= \frac{1}{2 \pi fC} \\ &= \frac{1}{(2) (3.14) (1 \text{ kHz}) (22.5 \text{ pF})} \\ &= \frac{1}{(6.28) (1 \times 10^3 \text{ Hz}) (22.5 \times 10^{-12} \text{ F})} \\ &= \frac{1}{1.41 \times 10^{-7} \text{ ohms}} \\ &= 7.07 \times 10^6 \text{ ohms} \\ &= 7.07 \text{ M}\Omega \end{aligned}$$

For the 4.5 pF capacitance:

$$\begin{aligned}
 X_C &= \frac{1}{2\pi fC} \\
 &= \frac{1}{(2)(3.14)(1 \text{ kHz})(4.5 \text{ pF})} \\
 &= \frac{1}{2.826 \times 10^{-8} \text{ ohms}} \\
 &= 35.4 \times 10^6 \text{ ohms} \\
 &= 35.4 \text{ M}\Omega
 \end{aligned}$$

The diagrams in Figure 8-6, through Figure 8-8, show other ways that capacitive sensors are used to measure both linear and angular position. Here, the distance between the capacitor plates remains constant, but the *effective* plate area of the capacitor changes. In one case, the position of one rectangular plate is moved in relation to the other stationary plate. In the other case, two cylinders form the plates of a cylindrical capacitor. As the inner cylinder plate moves within the outer cylinder plate, the capacitance changes.

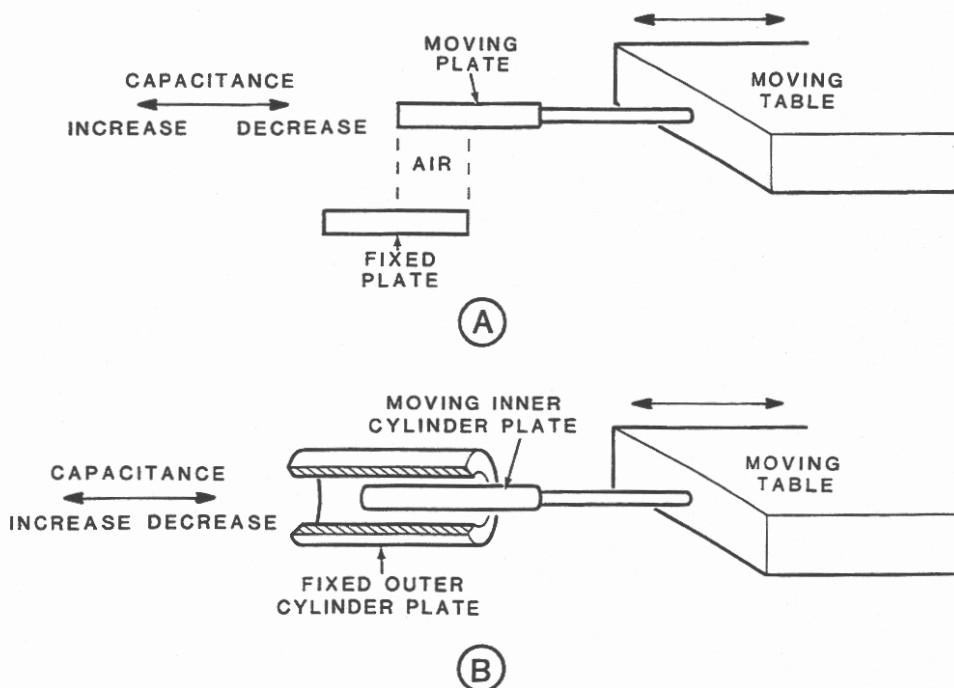


Figure 8-6

Measuring linear position by changing effective plate area of a capacitor.

In Figure 8-7 the capacitor plates are held stationary and the object is attached to a moveable dielectric. As the dielectric moves, the dielectric constant of the capacitor changes which changes the capacitance of the capacitor. Both rectangular and cylindrical capacitors are used in this application.

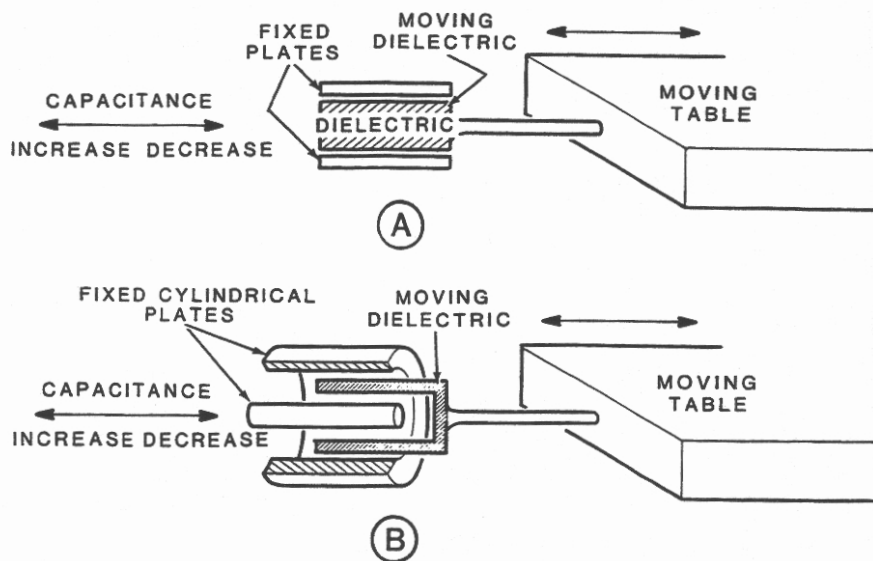


Figure 8-7

Measuring linear position by changing the dielectric constant of a capacitor.

Capacitive sensors are used to measure angular position as shown in Figure 8-8. Here, one plate is held stationary and the other is attached to the shaft of a rotating object. In Figure 8-8A, the distance between the plates changes as the shaft rotates. In Figures 8-8B and C, the effective plate area of the capacitor changes as one of the plates is rotated. You might recognize the capacitor in the middle as the familiar radio tuner-type capacitor.

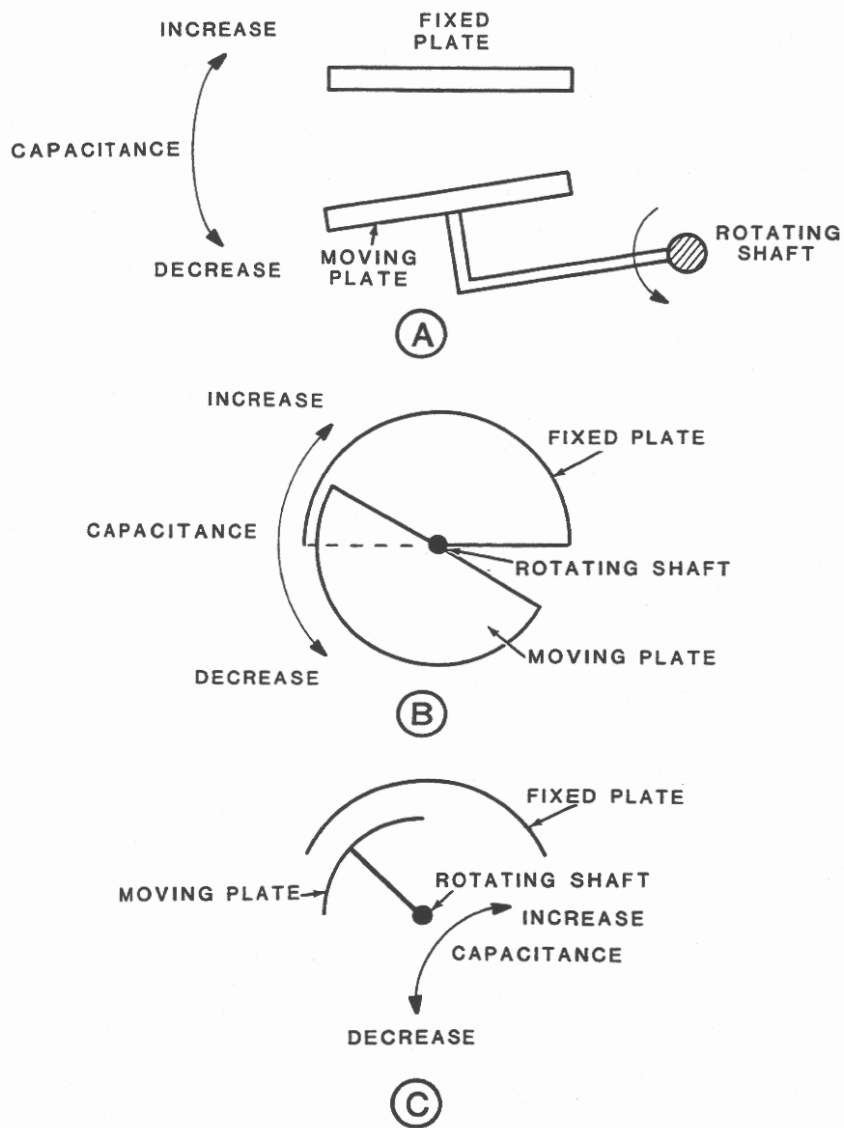


Figure 8-8

Capacitors are used to sense angular position by changing the effective plate area.



## SENSING PROXIMITY

A proximity sensor is one that responds to the near presence of an object. Capacitors can be used to sense the proximity of *conductive* objects as shown in Figure 8-9. With this arrangement, a conductive object is grounded and forms one plate of the capacitor. The other capacitor plate is contained in a sensing probe which is connected to an AC source. The dielectric material is the air gap between the object and probe. As the probe nears the object, the distance between the plates is reduced, resulting in an increase in capacitance.

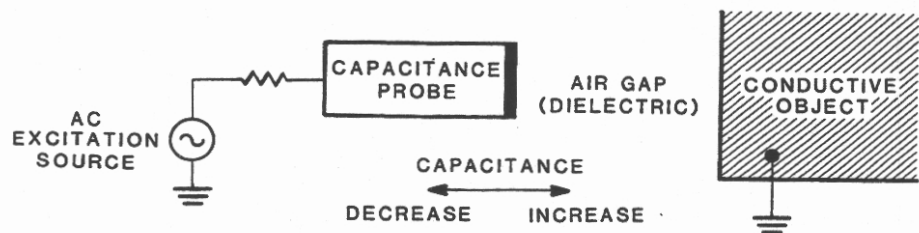


Figure 8-9

A capacitive proximity sensor.

Other capacitive proximity sensors, like the one shown in Figure 8-10, are commercially available to sense the presence of *both conductive and non-conductive* objects. They operate on the principle of a changing dielectric constant as an object comes in close proximity of the sensor. Most of these sensors are designed to operate on DC voltages anywhere between 5 V and 30 V. They generate a DC output that is proportional to the distance between the probe and the object. The **sensing distance** depends on the dielectric constant of the object material. Different sensors of this type have different ranges of materials that they can detect. For example, one sensor might be designed to detect steel, water, and wood, while another sensor is designed to detect glass, porcelain, and plastic.

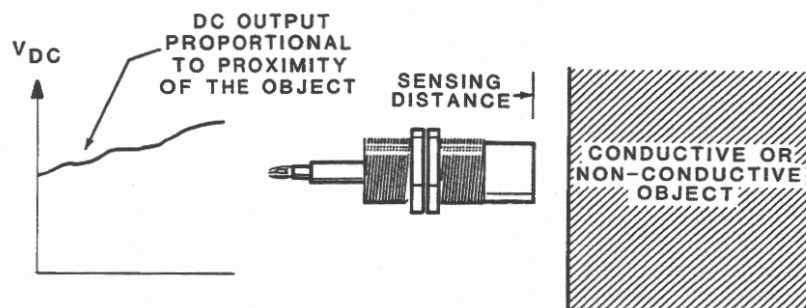


Figure 8-10

A typical commercial capacitive proximity sensor.

## Inductive Position and Proximity Sensors

Recall that an inductor in its most basic form is simply a coil of wire wrapped around a core as shown in Figure 8-11. The core material can be air, nickel, or a ferrous material such as iron, steel, permalloy, or supermalloy. An inductor can be used to sense position or proximity as a result of the following relationship:

$$L = \frac{(rN)^2}{9r + 10l} \mu_r$$

where:  $L$  is the inductance of the inductor in microhenries ( $\mu\text{H}$ ).

$r$  is the radius of the core in inches (in.).

$N$  is the number of coil turns.

$l$  is the effective length of the core in inches (in.).

$\mu_r$  is the relative permeability of the inductor core (see Table 8-2).

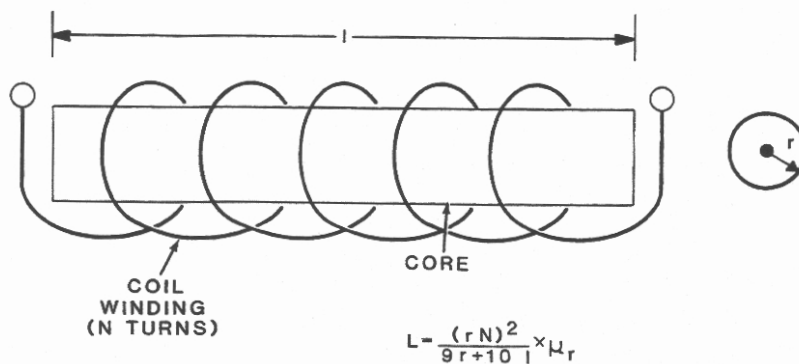


Figure 8-11  
Construction of a typical inductor.

Given an inductor with a particular core material, the only variable in the previous equation is the effective length of the core. The effective length of the core is that part of the core that is within the inductor coil.

| Material      | Relative Permeability, $\mu_r$ |
|---------------|--------------------------------|
| Vacuum        | 1                              |
| Air           | 1                              |
| Cobalt        | 170                            |
| Nickel        | 1,000                          |
| Iron          | 7,000                          |
| Permalloy*    | 100,000                        |
| Supermalloy** | 1,000,000                      |

\*Permalloy - 17% iron, 4% molybdenum, 79% nickel.  
 \*\*Supermalloy - 16% iron, 5% molybdenum, 79% nickel.

Table 8-2

Relative permeabilities for selected materials.

## SENSING POSITION

The diagram in Figure 8-12 shows how an inductor can be used to measure linear position. Here, an object is attached to the moveable core of an inductor. As the object moves, the effective length of the core changes, resulting in a change in inductance of the inductor.

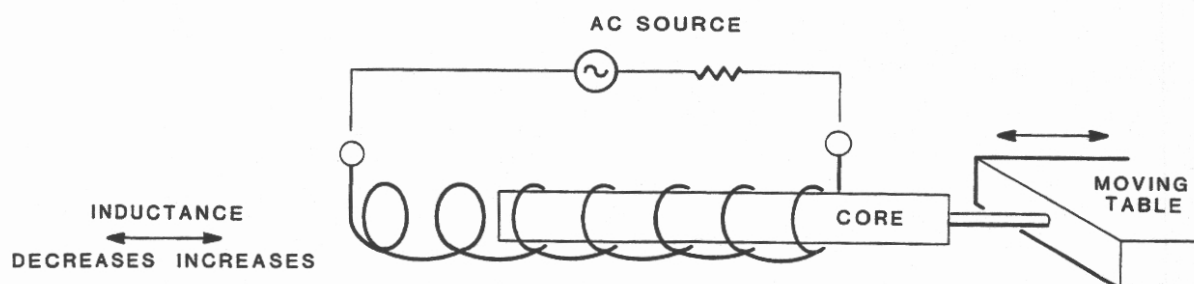


Figure 8-12

Measuring position using a simple inductor.

The resistance that an inductor offers to the flow of alternating current is called **inductive reactance**, or  $X_L$ , and can be calculated as follows:

$$X_L = 2 \pi f L$$

where:  $X_L$  is the inductive reactance in ohms.

$\pi$  is 3.14

$f$  is the frequency of the AC source in Hertz (Hz).

$L$  is the inductance of the inductor in Henrys.

So, by changing the effective length of the inductor core, the inductance of the inductor changes, which changes the inductive reactance according to the above relationship. This changes the voltage drop across the inductor by Ohm's law.

#### Example 8-3:

A 312 turn inductive sensor has a moveable iron core that is 4 inches long, and .5 inches in diameter.

- A. What is the inductance of the sensor when the core is positioned entirely within the coil?
- B. What is the inductance of the sensor when the core is positioned so that one-half of its length is within the coil?
- C. What is the inductive reactance of the sensor in parts A and B, when it is driven by a 1 kHz AC source?

#### Solution:

- A. The relative permeability for an iron core is 7000 from Table 8-2. Thus,

$$\begin{aligned}
 L &= \frac{(rN)^2}{9r + 10l} \times \mu_r \\
 &= \frac{[(.25\text{in}) (312)]^2}{9 (.25 \text{ in.}) + 10 (4 \text{ in.})} \times 7000 \\
 &= \frac{6084}{42.25} \times 7000 \\
 &= 1008000 \mu\text{H} \\
 &= 1.008 \text{ H}
 \end{aligned}$$

- B. When one-half of the core length is within the coil, the inductance of the sensor is:

$$\begin{aligned} L &= \frac{(rN)^2}{9r + 10l} \times \mu_r \\ &= \frac{[(.25\text{in}) (312)]^2}{9 (.25 \text{ in.}) + 10 (2 \text{ in.})} \times 7000 \\ &= \frac{6084}{22.25} \times 7000 \\ &= 1914067 \mu\text{H} \\ &= 1.914 \text{ H} \end{aligned}$$

- C. For the 1.008 H inductance in part A, the inductive reactance is:

$$\begin{aligned} X_L &= 2 \pi fL \\ &= (2)(3.14)(1 \text{ kHz})(1.008 \text{ H}) \\ &= 6330 \\ &= 6.33 \text{ k}\Omega \end{aligned}$$

For the 1.914 H inductance in part B, the inductive reactance is:

$$\begin{aligned} X_L &= 2 \pi fL \\ &= (2)(3.14)(1 \text{ kHz})(1.914 \text{ H}) \\ &= 12020 \\ &= 12.02 \text{ k}\Omega \end{aligned}$$

## THE LVDT

A special type of inductive position sensor is the **linear variable differential transformer**, or LVDT. The LVDT is one of the most common and accurate position sensors available. Commercial LVDTs are available that provide an indication of position within 1-2 micrometers.

Look at the LVDT in Figure 8-13. Notice that there are three windings: a single primary winding and two secondary windings around a single moveable core. In particular, observe that the two secondary windings are connected in series and wound in opposite directions.

An LVDT acts like a standard inductance transformer in that a voltage is induced from the primary winding onto the secondary windings. But, since the secondary windings are wound in opposite directions, the voltage induced onto one secondary winding is  $180^\circ$  out-of-phase with the voltage induced onto the other secondary winding.

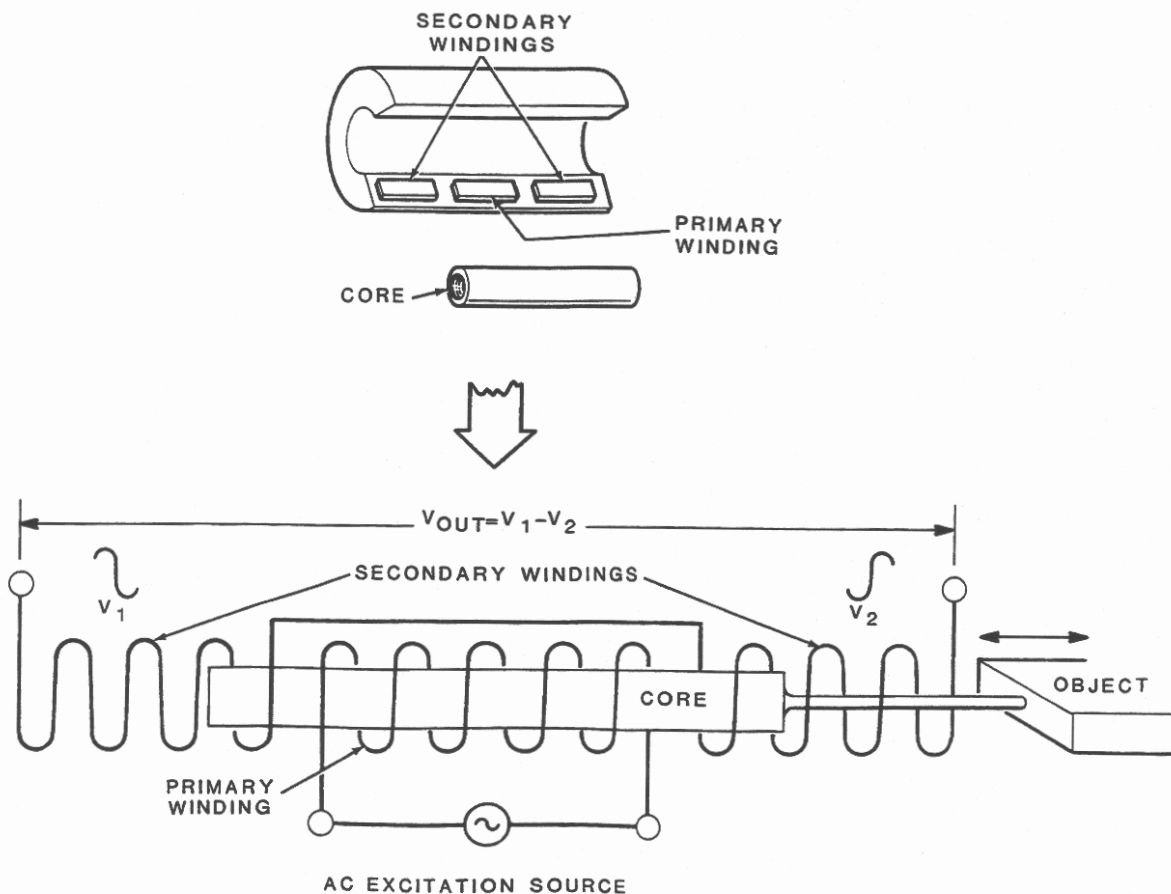


Figure 8-13  
A linear variable differential transformer, or LVDT.

When the core is centrally located, the voltages developed across the two secondary windings are equal in magnitude and 180° out-of-phase. As a result, the two voltages cancel each other and the output of the LVDT is 0 V as shown in Figure 8-14A. Now, when the core is moved in one direction or the other, the output of one secondary winding increases, while the other secondary winding output decreases. This is due to the different amount of magnetic flux linkage associated with the two secondary windings. As a result, the *combined* output generated by the LVDT increases as the core is displaced in either direction. Furthermore, the phase of the LVDT output indicates the direction of movement. In summary, the LVDT generates an output voltage whose magnitude is a function of the amount of core displacement, and whose phase is a function of the direction of core movement. This idea is illustrated by Figures 8-14B and 8-14C.

Remember, the LVDT generates an AC output which must be converted to DC prior to conversion to a digital signal. A peak detector signal conditioning circuit can be used for this purpose.

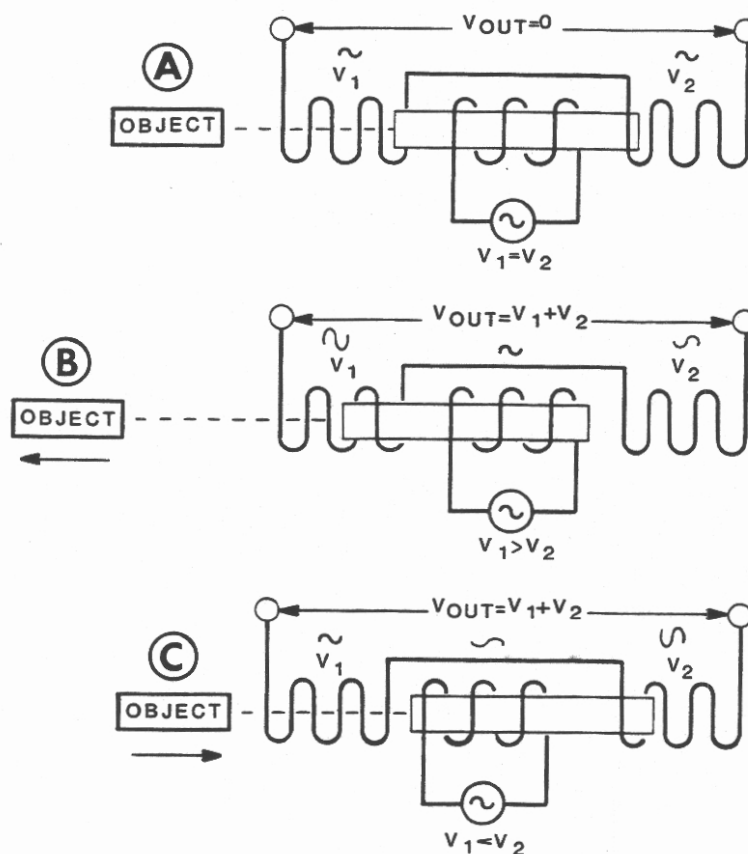


Figure 8-14

The core displacement of an LVDT causes the secondary outputs to be equal (A), or not equal (B and C).

## THE RVDT

The **rotary variable differential transformer**, or **RVDT**, shown in Figure 8-15 is the rotary counterpart of the LVDT. Like the LVDT, the RVDT produces a continuous voltage whose magnitude varies with the angular rotation of its protruding shaft. In operation, the RVDT shaft is attached directly to the object whose angular position is being measured. The shaft turns an internal ferromagnetic rotor core within stationary secondary stator windings, located above and below the rotor as shown. The combined differential output of the secondary windings is a voltage whose magnitude is proportional to the angular position of the shaft, and whose phase indicates the direction of shaft rotation. However, you should be aware that the rotation of the RVDT shaft is usually limited to  $\pm 90^\circ$ , with  $\pm 30^\circ$  being typical.

RVDTs are sometimes called **induction potentiometers**. Unlike resistive potentiometers, they do not require wiper-like contacts. Consequently, RVDTs do not exhibit as much mechanical wear as resistive potentiometers and offer minimum friction to the measurement process. Moreover, the input and output of an RVDT are electrically isolated from each other. Plus, like an LVDT, an RVDT generates an AC output that must be rectified to DC and conditioned prior to measurement by an A/D converter.

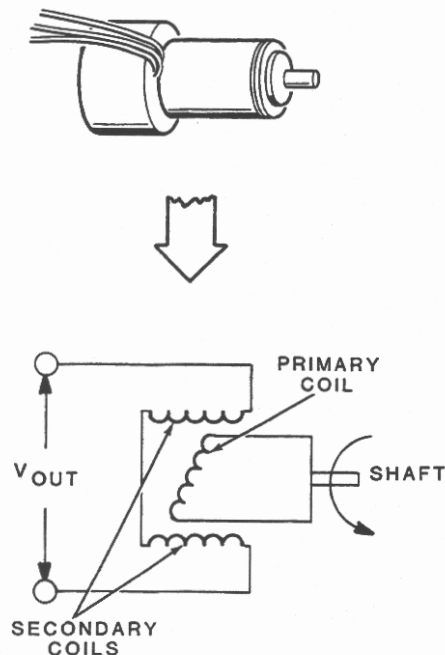


Figure 8-15

An RVDT is used to measure angular position.



## SENSING PROXIMITY

Inductive proximity sensors are used to sense the nearness of conductive materials. The idea is to put a small inductive coil winding on the inside of a sensor probe like the one shown in Figure 8-16. Observe how the coil forms part of an oscillator circuit. A DC source potential between 5 V and 30 V is applied to the sensor and internally chopped to produce an alternating current which energizes the oscillator circuit. This generates an alternating magnetic field around the coil.

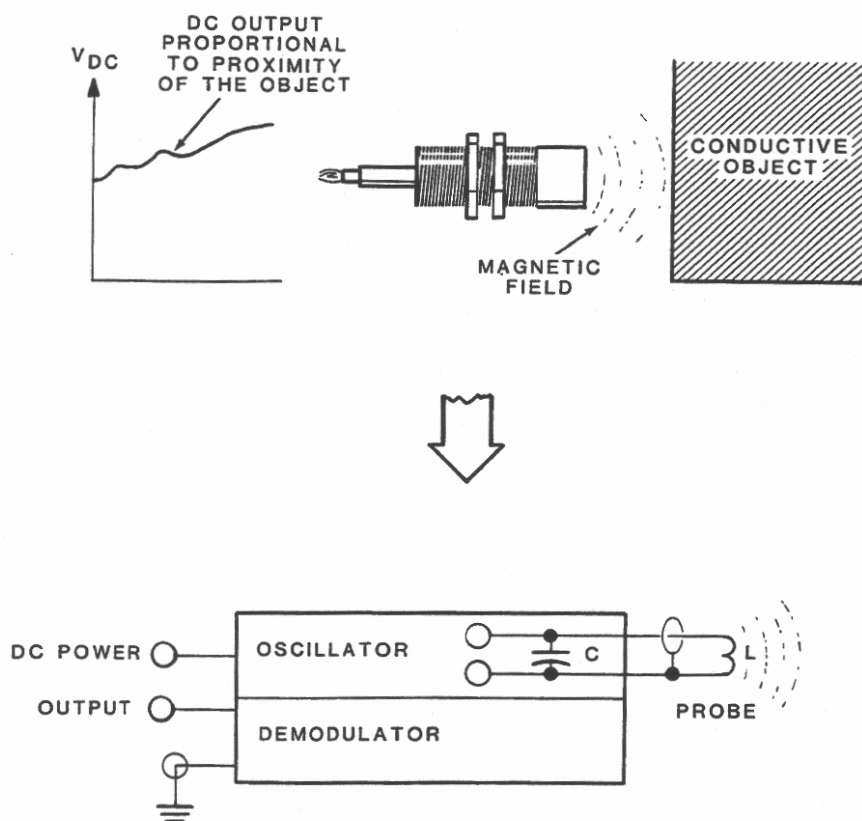


Figure 8-16

A typical commercial inductive proximity sensor.

When a metallic object enters the proximity of the magnetic field, eddy currents are created within the object through electromagnetic induction (Faraday's law). The eddy currents in the object create a magnetic field that opposes the original magnetic field produced by the coil (Lenz's law). As a result, voltages are induced back in the coil that oppose the original oscillating voltage. The net effect is a decrease in voltage across the coil that is proportional to the magnitude of the eddy currents within the metallic object. For this reason, inductive proximity sensors are sometimes called **eddy current probes**.

The amount of voltage decrease within the probe depends on the gap between the probe and the object, the object resistivity, the magnetic permeability properties of the object, and the oscillating frequency of the probe. However, with a given metallic object and oscillating frequency, the only variable is the gap between the probe and the object. Consequently, the voltage across the probe coil will change proportionally to the distance between the probe and the object. This change in AC voltage is demodulated and converted to a DC output.

Eddy current probes commonly have a range of an inch or more and a sensitivity of 100 to 200 millivolts per .001 inch of gap. As a result, they can be used to detect the presence of metallic objects as well as measure the distance to those objects. They are particularly useful for measuring vibration.

## Magnetic Position and Proximity Sensors

Magnetic position and proximity sensors are used to detect the presence of a magnetic field. The idea is to attach or embed a small magnet within a moving object as shown in Figure 8-17. Then, when the object passes by the magnetic sensor, the sensor generates a logic signal that indicates the presence of the object.

There are basically two simple devices that are commonly used to detect the presence of a magnetic field: the magnetic reed switch and the Hall-effect device. Let's take a closer look at each.

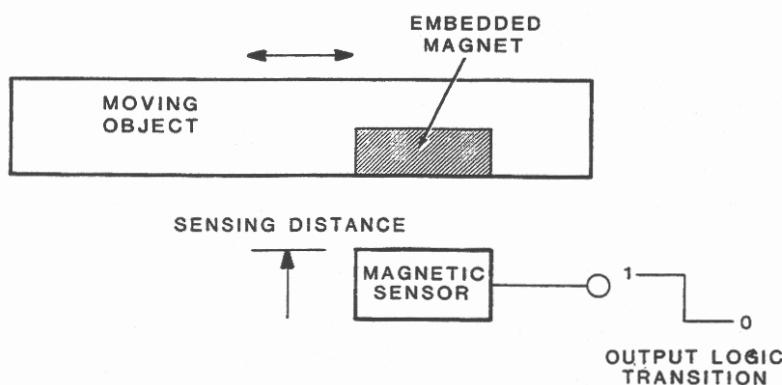


Figure 8-17

A magnetic sensor detects the presence of a small magnet attached or embedded into a moving object.

## THE MAGNETIC REED SWITCH

A magnetic reed switch is shown in Figure 8-18. As you can see, the switch consists of two ferromagnetic reed contacts enclosed in a glass tube. The entire device is approximately 1" in length and 1/8" in diameter. When a magnetic field is present, the reed contacts close as shown.

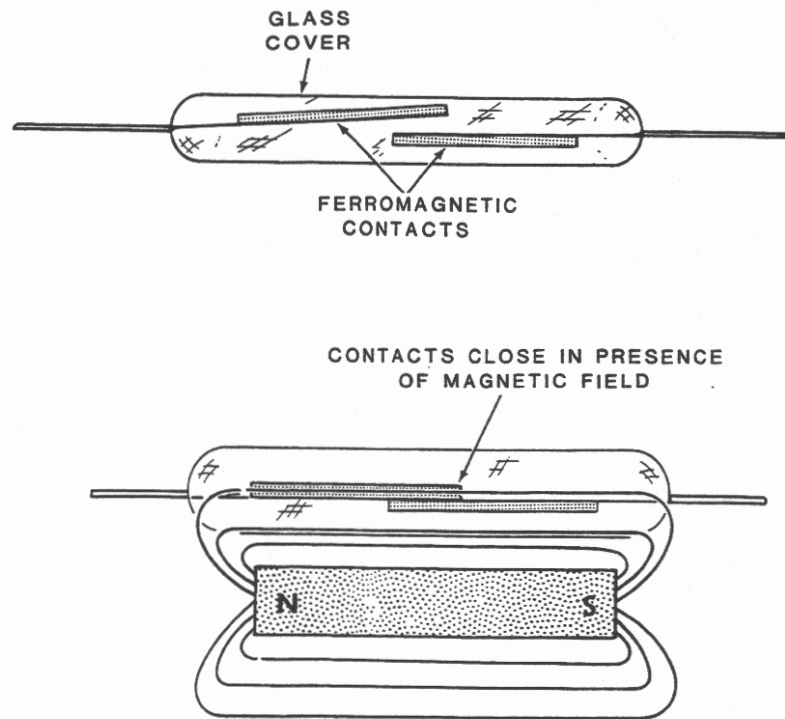


Figure 8-18

A magnetic reed switch consists of two magnetic reed contacts that close in the presence of a magnetic field.

The reed switch by itself does not generate a signal in the presence of a magnetic field—it must be connected as part of a pull-up or pull-down resistor circuit like those shown in Figure 8-19. Look at the pull-up resistor circuit in Figure 8-19A. When no magnetic field is present, the switch is open and no current flows from ground. Consequently, no voltage is dropped by the pull-up resistor and the full +5 V supply potential is seen at the circuit output. However, when a magnetic field is present the switch closes and conducts electron current from ground to the +5 V source. As a result, the entire +5 V supply potential is dropped across the pull-up resistor and a ground potential is seen at the output.

Since +5 V and ground are used to represent a logic 1 and 0, respectively, in computer circuits, the circuit output makes a logic transition from 1 to 0 when a magnetic field is present. This logic transition can be easily detected by a microprocessor.

The operation of the pull-down circuit in Figure 8-19B is just the opposite. It generates a logic transition from 0 to 1 in the presence of a magnetic field. Magnetic reed switches can also be used in this way to sense proximity.

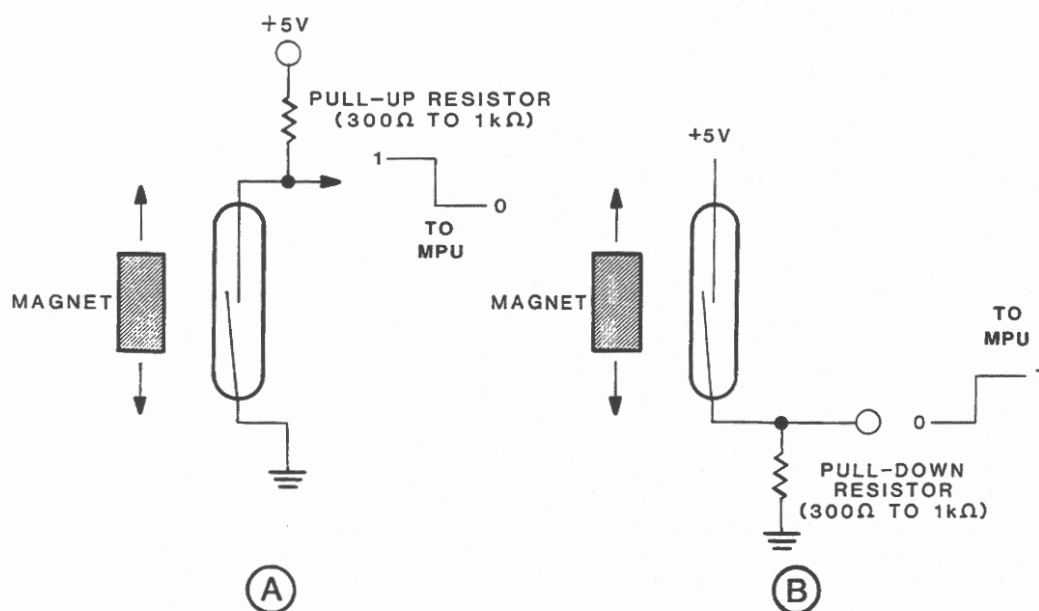


Figure 8-19

- A. A magnetic reed switch must be connected as part of a pull-up resistor, or
- B. Connected as part of a pull-down resistor circuit.

## HALL-EFFECT DEVICES

A Hall-effect device, shown in Figure 8-20, is a solid-state device that looks like a small-signal transistor. Since it is solid-state, it is more reliable and rugged than the glass-type magnetic reed switch.

As its name implies, the Hall-effect device operates on a "Hall-effect" principle. Hall-effect occurs as current flows through a semiconductor material that is under the influence of a magnetic field. As electrons or holes flow through the material, they experience a force whose direction depends on the polarity of the magnetic field.

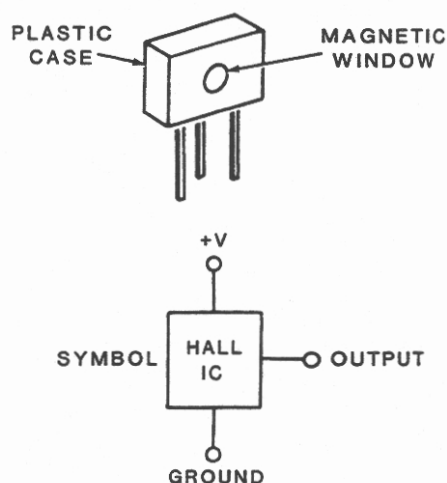


Figure 8-20

A Hall-effect device is a solid-state magnetic switch that looks like a small signal transistor.

The basic idea is that a Hall-effect device produces a Hall voltage that is proportional to the strength of an external magnetic field, and whose polarity is a function of the magnetic field polarity.

Hall-effect devices are used in the same way as magnetic reed switches for proximity sensing. A typical circuit connection is shown in Figure 8-21. Observe that a  $1\text{ k}\Omega$  pull-up resistor is inserted between the  $V_{cc}$  supply lead and the output lead of the device. Most Hall-effect devices are designed to operate on a  $V_{cc}$  potential anywhere between  $+5\text{ V}$  and  $+16\text{ V}$ . The third device lead is connected to ground. The output of this circuit will change state from a logic 0 to a logic 1 when the device is activated by the presence of a magnetic field of suitable strength.

You should be aware that a Hall-effect device requires a stronger magnetic field than a reed switch. Consequently, the proximity sensing distance of a Hall-effect device is less than that of a magnetic reed switch. Typical sensing distances range from less than  $1/16''$  to  $1/4''$  for Hall-effect devices, depending on the strength of the magnet being used to activate the device. Reed switches, on the other hand, can sense proximity up to  $1/2''$ .

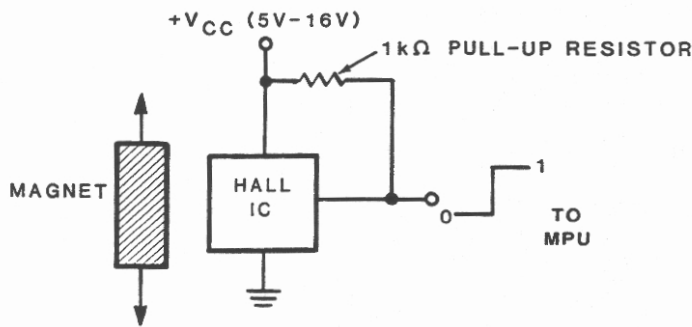


Figure 8-21

A typical circuit connection for a Hall-effect device.

## Sensing Position

Both linear and angular position can be measured using multiple magnetic reed switches, or Hall-effect devices as shown in Figure 8-22. In Figure 8-22A, a magnet is embedded into the bottom of a moving object such as a machine tool table. As the table moves laterally, the magnet passes over several separate reed switches or Hall-effect devices. Each sensor is located at a precise position, say every  $1/2''$ , so that the position of the table can be determined from the sequential activation of the sensors. Of course, the linear position resolution depends on the number and spacing of the magnetic sensors. Also, the sensors must be located close enough to the table to be within their given sensing range.

In Figure 8-22B, several magnetic reed switches or Hall-effect devices are positioned around a rotating shaft to measure angular position. In this application, a small permanent magnet is embedded into the shaft. As the shaft rotates, its angular position is determined by the sequential activation of the sensors. Again, the position resolution depends on the number of sensors that are used. If six sensors are used, the angular position of the shaft in Figure 8-22B could be resolved to  $60^\circ$  rather than  $90^\circ$ .

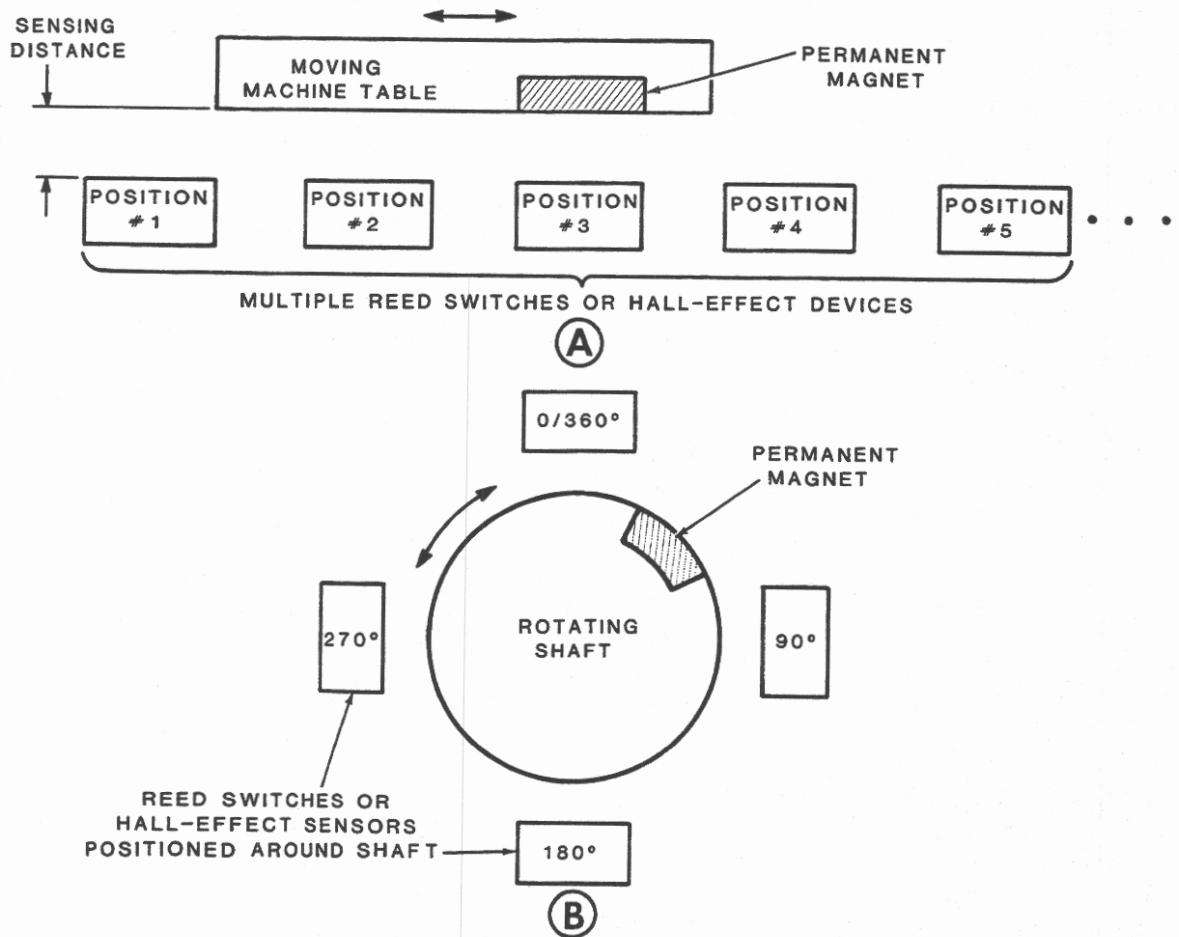


Figure 8-22

- A. Measuring linear position, and  
 B. Measuring angular position,  
 both using magnetic sensors.

The position and proximity sensors discussed in this section are summarized in Table 8-3.

Table 8-3  
Position/proximity sensor comparisons

| <u>TYPE OF DEVICE</u> | <u>ELECTRICAL VS. POSITION/MOTION CHARACTERISTICS</u>          | <u>CONSIDERATIONS</u>                                                                                                                                                                                                         |
|-----------------------|----------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Potentiometer         | Voltage output for wiper position.                             | Wiper must be attached directly or indirectly to object.<br>Can be used to measure angular or linear position.<br>Easiest way to determine position.                                                                          |
| Inductive             | Inductance changes with position.                              | Inductor core must be attached directly or indirectly to object.<br>Must be used as part of an AC bridge.<br>LVDT is a special inductive type transducer whose output must be detected and rectified.                         |
| Capacitive            | Capacitance changes with position.                             | One of the capacitor plates must be connected to the object.<br>Must be used as part of an AC bridge.                                                                                                                         |
| Magnetic              | Provides on/off switching in the presence of a magnetic field. | Economical and readily available.<br>Magnet(s) must be attached to the object.<br>Small sensing distance (1/16"-1/2").<br>Can be used to measure linear and angular position and velocity.<br>Doesn't require direct contact. |



## Self-Test Review

1. Calculate the resolution of a 100 turn potentiometer.
2. Suppose a voltage of 12 V is applied to the potentiometer in question 1. Determine the measurement accuracy in volts, of the potentiometer circuit.
3. Suppose the voltage follower op amp in Figure 8-3B represents a load of 100 M $\Omega$  to the potentiometer. Calculate the output voltage when the wiper is in the middle of its travel.
4. An air dielectric capacitive sensor has a moveable plate with a surface area of 5 in<sup>2</sup>. What is the capacitance of the sensor if the moveable plate is positioned .1 inch from the stationary plate?
5. What is the capacitive reactance of the sensor in question 4 when it is being driven with a 1000 Hz AC source?
6. A 200 turn inductive sensor has a moveable iron core that is 6 inches long and .25 inches in diameter. What is the inductance of the sensor when the core is positioned one-third of the way within the coil?
7. What is the inductive reactance of the sensor in question 6, when it is being driven with a 1 kHz AC source?
8. What is an LVDT and an RVDT, and how do their sensing applications differ?
9. Another name for an inductive proximity sensor is a/an \_\_\_\_\_.
10. List two types of magnetic position and proximity sensors.

\_\_\_\_\_.

\_\_\_\_\_.

## Answers

1. The resolution of a 100 turn potentiometer is:

$$\begin{aligned}\% \text{ Resolution} &= 100/N \\ &= 100/100 \text{ turns} \\ &= 1\%\end{aligned}$$

2. With a voltage of 12 V applied to the potentiometer in question 1, a 1% resolution translates to a measurement accuracy of:

$$1\% \text{ of } 12 \text{ V} = .12 \text{ V}$$

3. If the voltage follower op amp in Figure 8-3B represents a load of 100 M $\Omega$  to the potentiometer, the output voltage when the wiper is in the middle of its travels is:

$$\begin{aligned}V_{\text{out}} &= \left[ \frac{5 \text{ k}\Omega \parallel 100 \text{ M}\Omega}{5 \text{ k}\Omega + 5 \text{ k}\Omega \parallel 100 \text{ M}\Omega} \right] \times 10 \text{ V} \\ &= \left[ \frac{4.9975 \times 10^3}{9.9975 \times 10^3} \right] \times 10 \text{ V} \\ &= 4.998 \text{ V}\end{aligned}$$

$$\begin{aligned}4. \quad C &= \frac{.225 \text{ kA}}{d} \\ &= \frac{.225(1) (5 \text{ in}^2)}{.1 \text{ in}} \\ &= 11.25 \text{ pF}\end{aligned}$$

Note:  $k = 1$  for an air dielectric (Table 8-1).

$$\begin{aligned}
 5. \quad X_C &= \frac{1}{2\pi fC} \\
 &= \frac{1}{2(3.14)(1 \text{ kHz})(11.25 \text{ pF})} \\
 &= \frac{1}{2(3.14)(1 \times 10^3 \text{ Hz})(11.25 \times 10^{-12} \text{ F})} \\
 &= \frac{1}{7.065 \times 10^{-8}} \text{ ohms} \\
 &= 14.154 \times 10^6 \text{ ohms} \\
 &= 14.154 \text{ M}\Omega
 \end{aligned}$$

$$\begin{aligned}
 6. \quad L &= \frac{(rN)^2}{9r + 10l} \times \mu_r \\
 &= \frac{[(.125 \text{ in})(200)]^2}{9(.125 \text{ in}) + 10(2 \text{ in})} \times 7000 \\
 &= \frac{625}{21.125} \times 7000 \mu\text{H} \\
 &= 207100 \mu\text{H} \\
 &= .2071 \text{ H}
 \end{aligned}$$

$$\begin{aligned}
 7. \quad X_L &= 2\pi fL \\
 &= 2(3.14)(1 \text{ kHz})(.2071 \text{ H}) \\
 &= (6.28)(1 \times 10^3 \text{ Hz})(.2071 \text{ H}) \\
 &= 1.3 \text{ k}\Omega
 \end{aligned}$$

8. An LVDT is a linear variable differential transformer, and an RVDT is a rotary variable differential transformer. LVDTs are used to sense linear position while RVDTs are used to sense angular position.

9. Eddy Current Probe.

10. A. Magnetic reed switch.  
B. Hall-effect device.

## FORCE SENSING

The measurement of force and pressure is very common in industrial process control systems. A force acting on an object causes that object to change its shape, or to deform. By measuring the amount of deformation of the object, you can determine the amount of force acting on the object. A device called a strain gage (or "gauge") is commonly used to measure force. Strain gages use the principle that the resistance of a conductor changes in direct proportion to its length.

Pressure is defined as the force exerted by a liquid or gas. Pressure sensors and transducers measure this force indirectly by using displacement principles. Many of the resistive and inductive principles discussed earlier in this unit are used to measure pressure. In addition, solid-state pressure sensors are available that greatly simplify the measurement of pressure and associated signal conditioning for microprocessor-based data acquisition systems.

### Force

One of the more common measurements required in a mechanical process control system is force measurement. Process variables such as pressure, weight, and flow are all measured indirectly by measuring force.

When an adequate force is applied to a solid object, the object changes its physical dimensions, or deforms. The internal effect on the object caused by the force is called **stress**, and the resulting deformation is called **strain**.

In general, there are three types of stress/strain situations: **compression stress/strain**, **tensile stress/strain**, and **shear stress/strain**. Compression and tensile stress/strain cause an object to change its length as illustrated in Figure 8-23A and B. A compression stress applied to an object causes that object to compress. Consequently, the resulting strain is a decrease in the object's length. A tensile stress is just the opposite of a compression stress. Tensile stress on an object causes the object to become longer. Thus, the resulting strain is an increase in the object's length.

Another type of strain is illustrated in Figure 8-23C, using a book as an example. If you applied a horizontal force to a thick book setting on a table as shown, you would see the results of shear strain as a change in the angle that the book pages make in relationship to the table. As you can see, shear stress results in angular distortion. Since shear strain is not as common in the industrial environment, we will concentrate on compression and tensile strain.

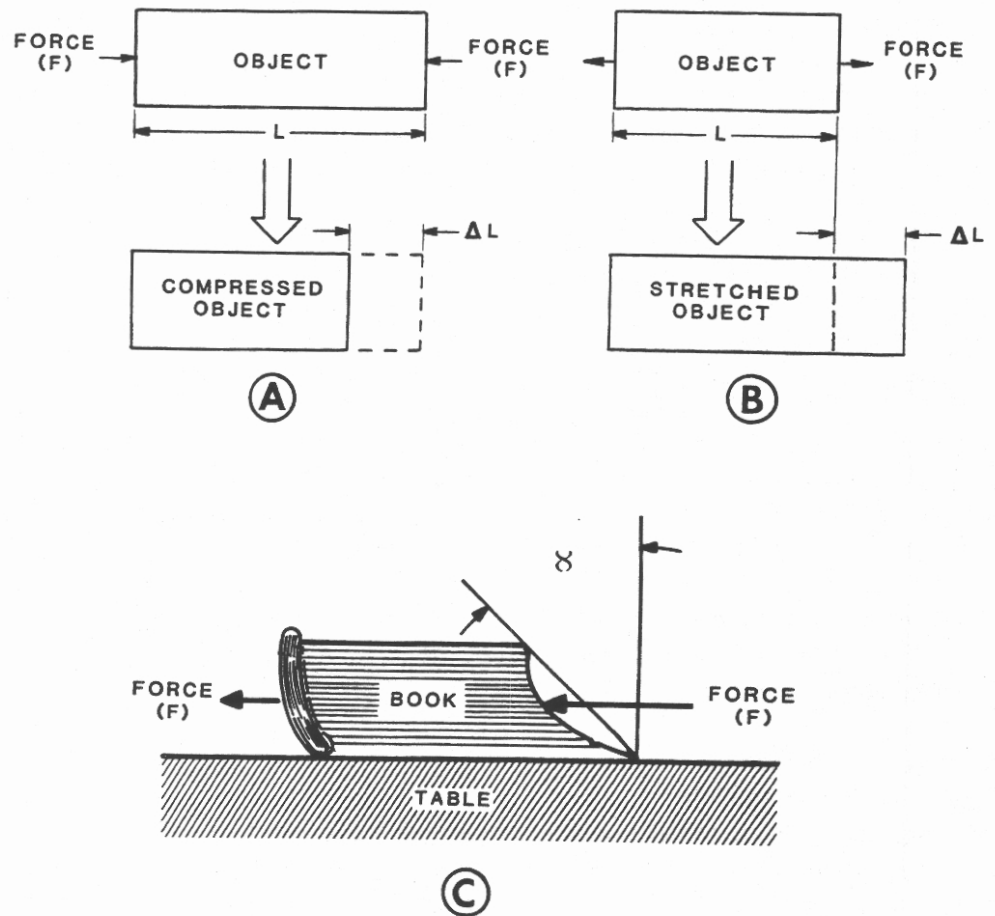


Figure 8-23

- A. Compression strain.
- B. Tensile strain.
- C. Shear strain.

The amount of compression or tensile strain on an object is defined as the change in length,  $\Delta L$ , of an object due to an applied force, divided by the original length,  $L$ , of the object. Consequently, you can calculate compression and tensile strain as follows:

$$\text{Strain} = \Delta L / L$$

where:  $\Delta L$  is the change in the object's length caused by the applied force.

$L$  is the original length of the object.

Strain has a positive (+) value for tensile stress, since the length of the object increases, and a negative value (−) for compression stress, since the length of the object decreases. You can determine from this definition that strain is a unitless quantity, since inches will cancel inches, meters will cancel meters, and so on.

Strains are usually expressed in micro, or  $\mu$ , quantities, where  $\mu$  is a prefix meaning  $10^{-6}$ . Thus, a strain of 1000  $\mu$  is equivalent to  $1000 \times 10^{-6}$ , or .001. To express a strain in micro quantities, you simply multiply the strain by  $\frac{1}{10^{-6}}$ , or  $10^6$ .

As mentioned earlier, to measure the amount of force applied to an object, you must measure the amount of deformation, or strain, resulting from the applied force. Sensing devices used to measure strain are called **strain gages**. The most common types of strain gages are resistive and semiconductor strain gages.

## Resistive Strain Gages

Resistive strain gages are by far the most popular gage used in the industry. They operate on the principle that the resistance of a conductor changes in proportion to its length. When a conductor is stretched, its resistance increases. Conversely, its resistance decreases when it is compressed. In fact, the change in resistance,  $\Delta R$ , of a resistive strain gage under stress can be approximated as follows:

$$\Delta R = 2R_{\text{old}} \times \text{Strain}$$

where:  $\Delta R$  is the change in resistance of the conductor in ohms.

$R_{\text{old}}$  is the original resistance of the conductor in ohms.

$$\text{Strain} = \Delta L / L$$

**Example 8-4:**

Suppose the nominal unstrained resistance of a resistive strain gage is  $120\ \Omega$ . A force is applied to the strain gage that results in a strain of  $1000\ \mu$ . Calculate the resulting change in resistance of the strain gage.

**Solution:**

Using the above relationship, the change in resistance due to the applied force is:

$$\begin{aligned}\Delta R &= 2R_{\text{old}} \times \text{Strain} \\ &= 2(120\ \Omega)(1000\ \mu) \\ &= (240)(1000 \times 10^{-6}) \text{ ohms} \\ &= .24 \text{ ohms}\end{aligned}$$

**CONSTRUCTION AND OPERATION**

The two most common resistive strain gages are the wire bonded and foil bonded strain gages shown in Figure 8-24. The wire bonded strain gage consists of a small wire cemented to a thin paper or plastic film, in an "accordion" type fashion as shown in Figure 8-24A.

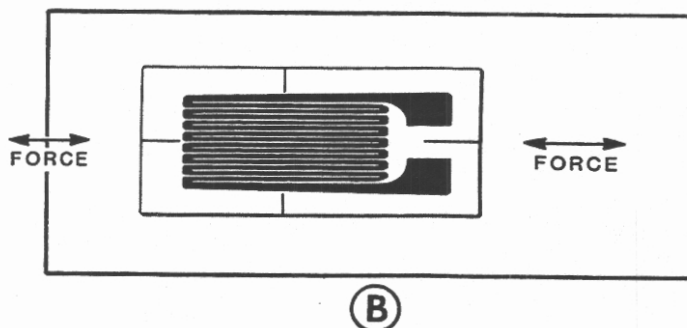
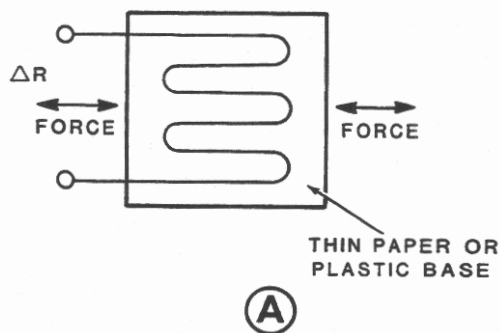


Figure 8-24

A. A wire bonded strain gage.

B. A foil bonded strain gage.

The foil bonded strain gage, sometimes called a **metal-film gage**, is made using a printed circuit process. This process uses a conductive alloy that is photoetched onto a substrate called a **carrier matrix**. Again, several conductor loops are provided in an accordion arrangement so that the nominal, unstrained resistance of the gage is sufficiently large. The total length of all the loops may range from 1/4 inch to over 12 inches, resulting in a nominal unstrained resistance. This resistance can range anywhere from between 50  $\Omega$  to 5000  $\Omega$ . Several shapes and sizes of foil bonded strain gages are available, some of which are pictured in Figure 8-25.

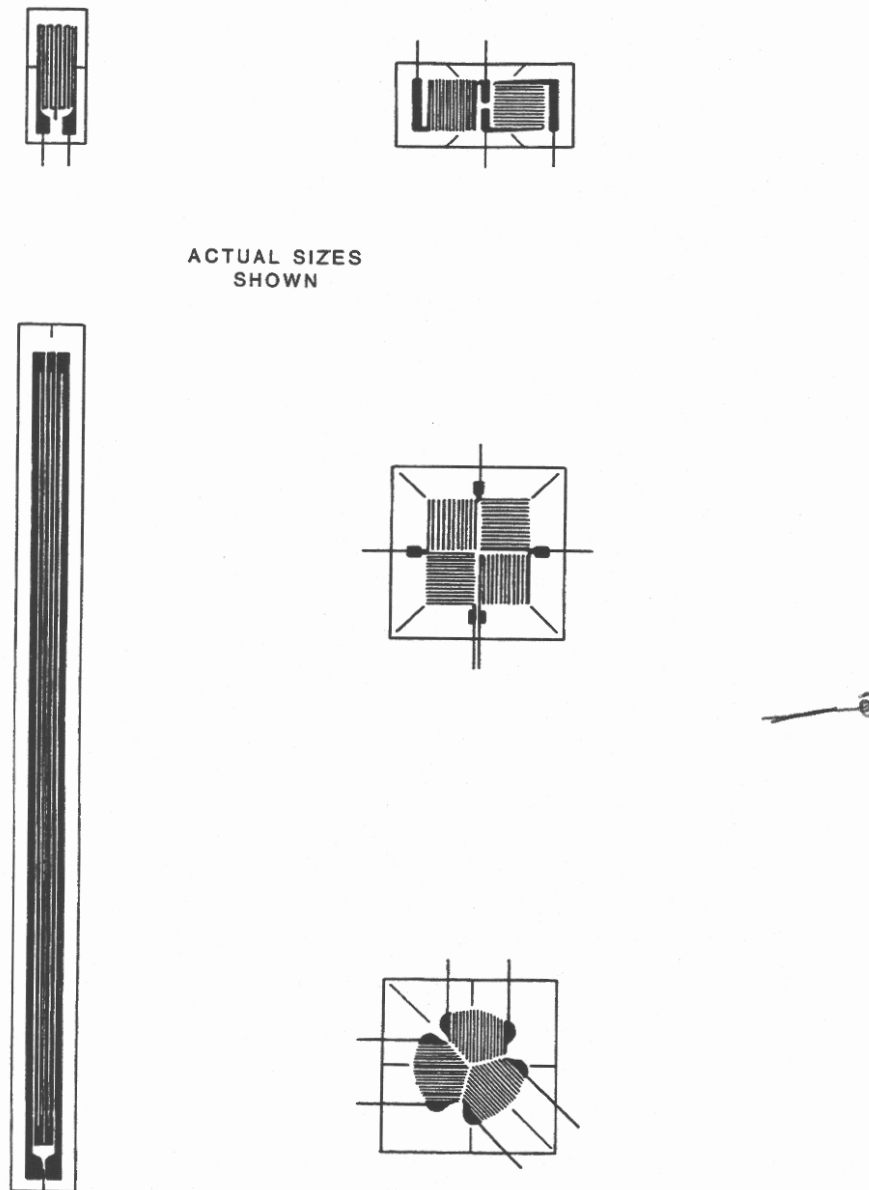


Figure 8-25

Foil strain gages are available in several shapes and sizes.



All resistive strain gages are rated by a sensitivity factor called gage factor, or GF. Gage factor is defined as follows:

$$GF = \frac{\Delta R/R_{\text{nominal}}}{\text{Strain}}$$

where: GF is the gage factor of the strain gage.

$\Delta R$  is the change in resistance due to strain in ohms.

$R_{\text{nominal}}$  is the nominal unstrained resistance of the gage in ohms.

$$\text{Strain} = \Delta L/L$$

#### Example 8-5:

Calculate the gage factor of the strain gage in Example 8-4.

#### Solution:

The nominal unstrained resistance of the strain gage in Example 8-4 is 120  $\Omega$ . You calculated that a strain of 1000  $\mu$  produced a resistance change of .24  $\Omega$ . Using these values directly in the gage factor equation gives you:

$$\begin{aligned} GF &= \frac{\Delta R/R_{\text{nominal}}}{\text{Strain}} \\ &= \frac{.24 \Omega / 120 \Omega}{1000 \times 10^{-6}} \\ &= 2 \end{aligned}$$

Since gage factor is an indication of the strain gage's sensitivity, the higher the gage factor the more sensitive the gage. Gage factors for resistive strain gages range from 2 to 10, depending on the material used for the resistive strain element.

In operation, the strain gage must be cemented to the object under stress as shown in Figure 8-26. It must be positioned so that the length of the coil turns are at 90° angles to the applied force, as shown. A force applied perpendicular to the length of the coil turns will stretch or compress the resistive element, resulting in a resistance change. If the force were applied in the same direction as the length of the coil turns, any compression or tensile stress on the gage would simply result in an accordion action, with no resistance change.

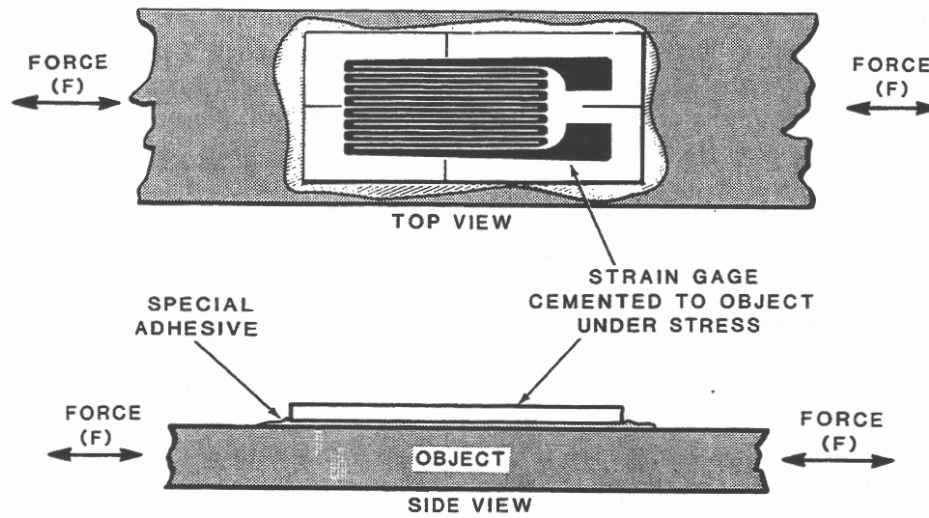


Figure 8-26

A strain gage must be cemented to the object under stress.

Two strain gages are often attached to the top and bottom of a cantilever arm as shown in Figure 8-27. When a downward force is exerted on the cantilever, the top gage is stretched due to a tensile stress, and the bottom gage is compressed due to a compression stress. The resistance change of the top gage is directly proportional to the applied force, and the resistance of the bottom gage is inversely proportional to the force. Conversely, an upward force causes the opposite effect.

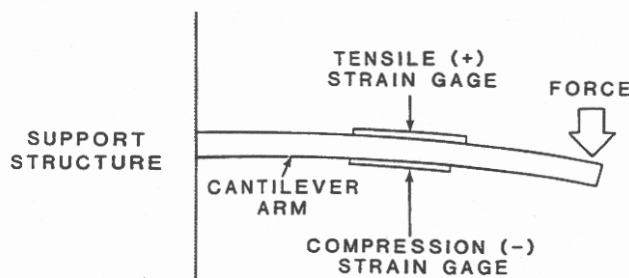


Figure 8-27

Two strain gages are used to measure the bending force acting on a cantilever arm.

## CHARACTERISTICS

As you learned earlier, the nominal strain gage resistance values range from 50  $\Omega$  to 5000  $\Omega$ . However, the most common values used in the industry are 120  $\Omega$  and 350  $\Omega$ . This nominal unstrained resistance depends on the gage length of the gage. Gage length is the width of the coil turns, and can range from .01 inch to over 4 inches.

Resistive strain gages are often preferred over semiconductor strain gages because they are very linear. In other words, their resistance is a linear function of the applied force. In addition, resistive strain gages are not affected as much by temperature variations as semiconductor strain gages. However, both the gage resistance and gage factor will change slightly with temperature changes. For this reason, temperature compensation might be required for critical sensing applications. This topic will be discussed shortly.

## APPLICATION CIRCUITS

The most common circuit used with a resistive strain gage is the DC Wheatstone bridge. The strain gage is placed in one leg of a Wheatstone bridge as shown in Figure 8-28. This is called a 1/4 bridge configuration, since the strain gage device makes-up one of the four legs of the bridge. Now, the voltage output,  $V_{out}$ , of the Wheatstone bridge is calculated as follows:

$$V_{out} = \left[ \frac{R_2}{R_1 + R_2} - \frac{R_{cal}}{R_g + R_{cal}} \right] \times V_{in}$$

Here,  $R_1$ ,  $R_2$ , and  $R_{cal}$  are fixed and  $R_g$  is the strain gage resistance. If  $V_{in}$  is held constant, the bridge output will be directly proportional to the strain gage resistance.

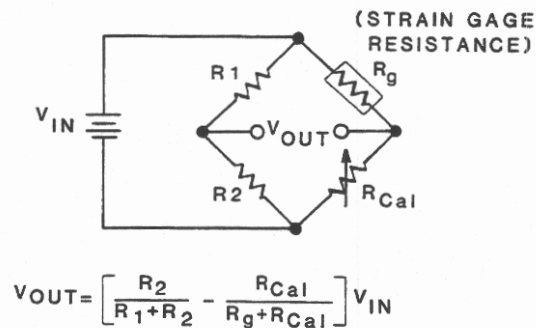


Figure 8-28

A strain gage placed in one leg of a DC Wheatstone bridge, in a 1/4 bridge configuration.

**Example 8-6:**

Suppose  $R_1$  and  $R_2$  in Figure 8-28 each have a fixed resistance value of 10 k $\Omega$  and the source voltage is 12 v.

- A. What must  $R_{cal}$  be adjusted to in order to balance the bridge under an unstrained condition, using the strain gage in Example 8-4?
- B. Assuming the bridge is balanced for an unstrained condition, what is the bridge output for a tensile strain of 1000  $\mu$ ?

**Solution:**

- A. Since  $R_1$  and  $R_2$  are equal, the value of  $R_{cal}$  must equal the strain gage resistance to balance the bridge. (Why?) Thus, using the 120 ohm strain gage from Example 8-4,  $R_{cal}$  must be adjusted to 120 ohms to achieve a balanced condition.
- B. From Example 8-4, a 1000  $\mu$  strain results in a resistance change of .24  $\Omega$ . Since the strain is a tensile strain, the strain gage resistance increases by .24  $\Omega$ . As a result, the strained resistance of the gage is 120  $\Omega$  + .24  $\Omega$ , or 120.24  $\Omega$ . Using this value in the previous voltage equations gives you:

$$\begin{aligned}
 V_{out} &= \left( \frac{R_2}{R_1 + R_2} - \frac{R_{cal}}{R_g + R_{cal}} \right) \times V_{in} \\
 &= \left[ \frac{10 \text{ k}\Omega}{10 \text{ k}\Omega + 10 \text{ k}\Omega} - \frac{120 \Omega}{120.24 \Omega + 120 \Omega} \right] \times 12 \text{ V} \\
 &= .006 \text{ V} \\
 &= 6 \text{ mV}
 \end{aligned}$$

In Example 8-6, the bridge was balanced for an unstrained condition. Once balanced, the bridge output is directly proportional to the amount of force applied to the strain gage. What will the bridge output be for an equal amount of compression strain? (Think about it!)

When using a microprocessor, the bridge does not have to be balanced for the unstrained condition. The microprocessor “reads” the bridge output for the unstrained condition, and then reads the output for a strained condition. Knowing the input voltage to the bridge, the computer can then calculate a **difference ratio voltage**,  $V_r$ , as follows:

$$V_r = \left[ \frac{V_{out}}{V_{in}} \right]_{unstrained} - \left[ \frac{V_{out}}{V_{in}} \right]_{strained}$$

Now if the computer knows the gage factor of the strain gage, it can calculate strain, using the following relationship:

$$\text{Strain} = \frac{-4V_r}{GF(1 + 2V_p)}$$

#### Example 8-7:

Suppose a microprocessor measures an unstrained output of 100 mV from the bridge in Example 8-6, and a strained output of 94 mV. Assume the bridge is being supplied from a +12 V source and the microprocessor is programmed with a gage factor of 2.

- A. Calculate the strain.
- B. Is the strain the result of a tensile or compression stress?

#### Solution:

- A. First, the MPU must calculate the difference ratio voltage,  $V_r$ , as follows:

$$\begin{aligned} V_r &= \left[ \frac{V_{out}}{V_{in}} \right]_{unstrained} - \left[ \frac{V_{out}}{V_{in}} \right]_{strained} \\ &= [100 \text{ mV}/12\text{V}] - [94 \text{ mV}/12 \text{ V}] \\ &= .5 \text{ mV} \end{aligned}$$

Next, the MPU must calculate the strain as follows:

$$\begin{aligned}\text{Strain} &= \frac{-4V_r}{GF(1 + 2V_r)} \\ &= \frac{-4(.5 \text{ mV})}{2[1 + 2(.5 \text{ mV})]} \\ &= \frac{-2 \text{ mV}}{2[1 + 1 \text{ mV}]} \\ &= \frac{-2 \text{ mV}}{2002 \text{ mV}} \\ &= -.0000999 \\ &= -999 \times 10^{-6} \\ &= -999 \text{ micros}\end{aligned}$$

- B. The previous strain calculation in A resulted in a negative value. Consequently, the strain is due to compression stress.

Using this procedure, the bridge does not have to be balanced for the unstrained condition, and the strain gage does not have to be extremely linear. The MPU is simply programmed to perform the previous calculations when a strain measurement is required.

One thing you should note in Examples 8-6 and 8-7 is that the bridge output changes very little from the unstrained condition. The change is so small that the bridge output must be amplified, using an instrumentation amplifier circuit as shown in Figure 8-29. As a review, an instrumentation amplifier is a signal conditioning circuit that amplifies the bridge output to make it compatible with the input of an A/D converter. In addition, the amplifier cancels out any noise that might be induced on the output signal.

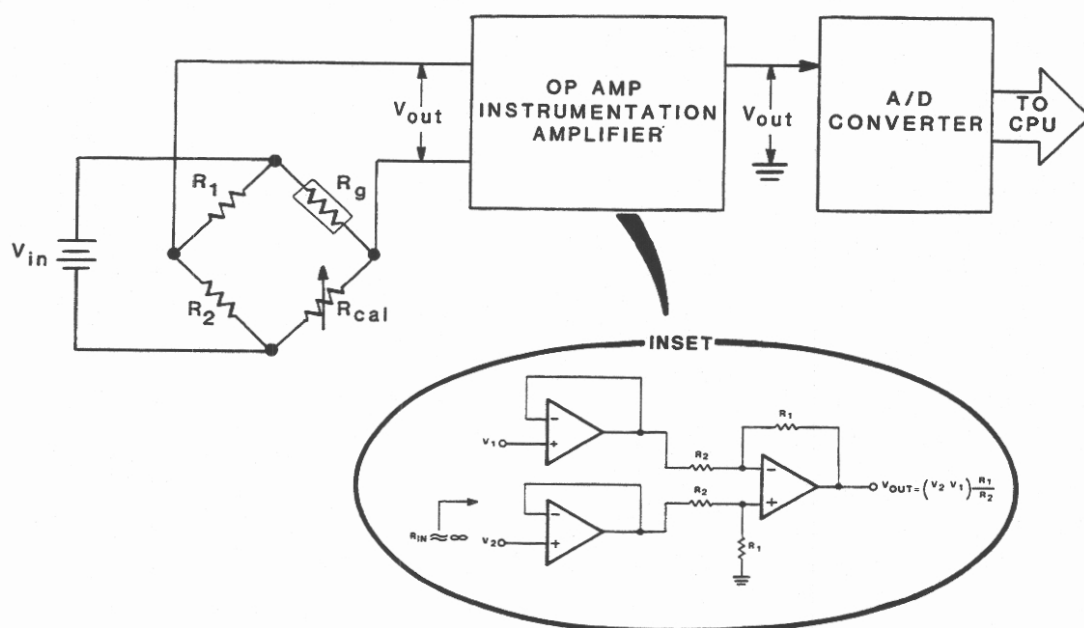


Figure 8-29

The Wheatstone bridge output must be conditioned using an instrumentation amplifier.

Often, the strain gage device must be located several feet from the bridge circuit. When this is the case, you must consider three things: **lead resistance**, **temperature effects** on the leads, and **noise**. The diagram in Figure 8-30 illustrates the solution to all three problems. First, if the lead wire is #18 or #20 AWG, the error due to lead resistance is less than 1% for 100 feet of wire using a 120 ohm strain gage. Of course, the lead resistance error can be reduced even further by using larger lead wires or a strain gage with a larger nominal resistance. For instance, using 100 feet of AWG #18 wire and a 350-ohm strain gage reduces the lead resistance error to less than .2%.

Second, temperature effects on the leads can be cancelled-out by using the three-wire arrangement shown. Notice the top and bottom leads are in two adjacent legs of the bridge. Thus, any change in resistance due to temperature in these two leads is cancelled-out by the action of the bridge circuit. The middle lead resistance does not affect the bridge output, since ideally it does not conduct any current when the bridge is balanced.

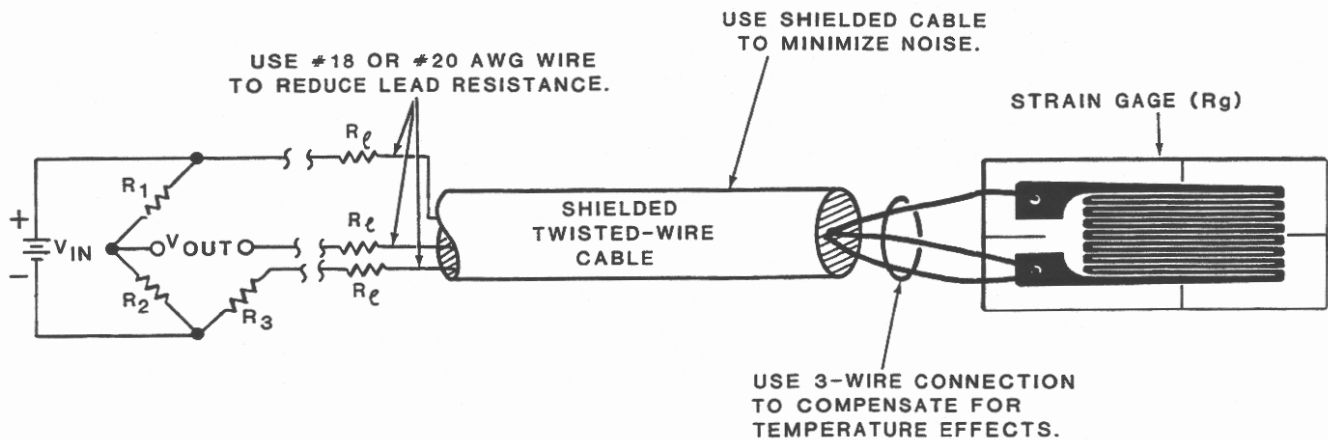


Figure 8-30

Three problems when locating strain gages some distance from the bridge circuit are: lead resistance, temperature effects on the leads, and noise.

Finally, you can minimize the amount of electrical noise induced onto the leads by using a shielded twisted cable, as shown.

There is one final thing you must consider when using a strain gage: the affect of temperature on the gage itself. Remember, that temperature affects the resistance of any conductor. The resistance of a strain gage is no exception. Strain gage resistance values are specified by the manufacturer at room temperature. If the sensing environment is different from room temperature, you must compensate for temperature by using two strain gages as shown in Figure 8-31.

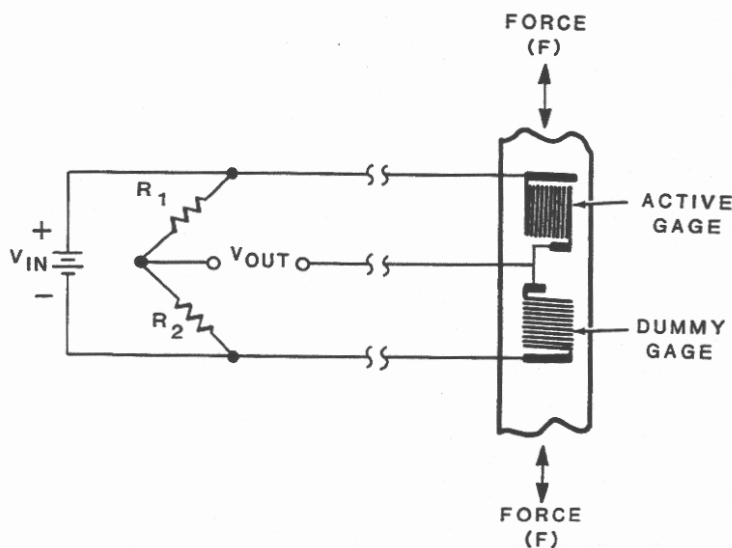


Figure 8-31

A dummy gage must be used to compensate for temperature variations.



Here, one strain gage is a dummy gage and the other is the active strain gage. The dummy gage is mounted on the object adjacent to the active gage, but positioned  $90^\circ$  in relation to the active gage. Thus, forces affecting the active gage do not affect the dummy gage, and vice-versa. However, both gages are affected equally by temperature changes. The two gages must have the same nominal resistance, and be connected into adjacent legs of the Wheatstone bridge as shown. With this arrangement, the effect of any common resistance changes due to temperature are cancelled-out. However, changes in force only affect the active gage, resulting in a proportional output voltage change from the bridge circuit. Two-element foil strain gages like the one shown are commercially available for this purpose. These components have two strain gage devices bonded on the same carrier matrix, but positioned  $90^\circ$  to each other.

## Semiconductor Strain Gages

Semiconductor strain gages operate on a piezoresistive principle. That is, the resistance of the semiconductor crystal changes with an applied compression or tensile stress.

### CONSTRUCTION AND OPERATION

Semiconductor strain gages are manufactured by bonding a band, or strip, of semiconductor material onto an insulative backing, as shown in Figure 8-32. The semiconductor device can be as small as .25 millimeters wide and .5 millimeters long. Two leads are attached to the device as shown. The entire strain gage assembly must be cemented to the object under stress, in the same way as a foil strain gage. The gage must be positioned as shown, so that the applied force either stretches or compresses the length of the semiconductor device.

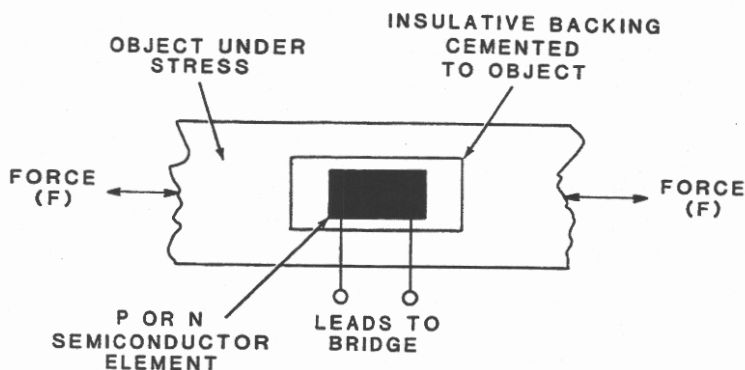


Figure 8-32  
A semiconductor strain gage.

The gage device can be either a P-type or N-type semiconductor. A P-type semiconductor has a positive temperature coefficient of resistance, while an N-type semiconductor has a negative temperature coefficient. The resistance range and sensitivity of the gage depend on the amount of doping that occurs during the manufacturing of the device.

## CHARACTERISTICS

The biggest advantage of a semiconductor strain gage is its extreme sensitivity. Semiconductor strain gages are up to 100 times more sensitive than foil strain gages. For instance, a typical gage factor for a semiconductor strain gage is 150 as compared to 2 for a foil gage. Semiconductor strain gages are available to measure strains up to 10,000  $\mu$ . However, 3000  $\mu$  is the practical strain limit of most commercial devices, with a typical resistance range of 50  $\Omega$  to 1000  $\Omega$ .

The two biggest disadvantages of a semiconductor strain gage are its nonlinearity and extreme sensitivity to temperature. The best way to compensate for both nonlinearity and temperature is to use software compensation. Memory look-up tables are commonly used to compensate for nonlinearity. Temperature compensation requires the computer to measure the actual environment temperature using a temperature sensing device. The strain gage output is then adjusted according to the temperature measurement.

## APPLICATION CIRCUITS

Semiconductor strain gage devices are usually placed in a Wheatstone bridge circuit like the foil strain gages discussed previously. The same precautions should be taken for lead resistance, noise, and temperature compensation. One advantage of using a semiconductor strain gage over a foil gage in a Wheatstone bridge is that the bridge output does not need to be amplified because of the high sensitivity of the device.

### Example 8-8:

Suppose a semiconductor strain gage has a nominal resistance of 120  $\Omega$  and a gage factor of 100. What is the change in resistance of the gage for a strain of 1000  $\mu$ ?

**Solution:**

- A. You can solve for the change in gage resistance,  $\Delta R$ , using the gage factor equation as follows:

$$GF = \frac{\Delta R / R_{\text{nominal}}}{\text{Strain}}$$

Solving for  $\Delta R$ , you get?

$$\Delta R = GF \times \text{Strain} \times R_{\text{nominal}}$$

Now, substituting the given values, gives you:

$$\begin{aligned}\Delta R &= 100 \times 1000 \mu \times 120 \Omega \\ &= 100 \times (1000 \times 10^{-6}) \times 120 \Omega \\ &= 12 \Omega\end{aligned}$$

## Piezoelectric Strain Gages

Piezoelectric strain gages generate a changing voltage that is proportional to an applied force. These devices are made from piezoelectric crystals like your stereo phonograph needle. An applied force causes the crystal to deform and generate a voltage, due to the piezoelectric principle. However, the voltage generated by the crystal is only present when the force is dynamic, or changing. For this reason, piezoelectric strain gages are not used to measure static, or constant, forces. Rather, they are useful for measuring dynamic forces such as acceleration, vibration, and shock.

## Load Cells

A load cell is a device that uses strain gages to measure weight. Commercial load cells like those pictured in Figure 8-33 are available to measure several pounds or several million pounds.

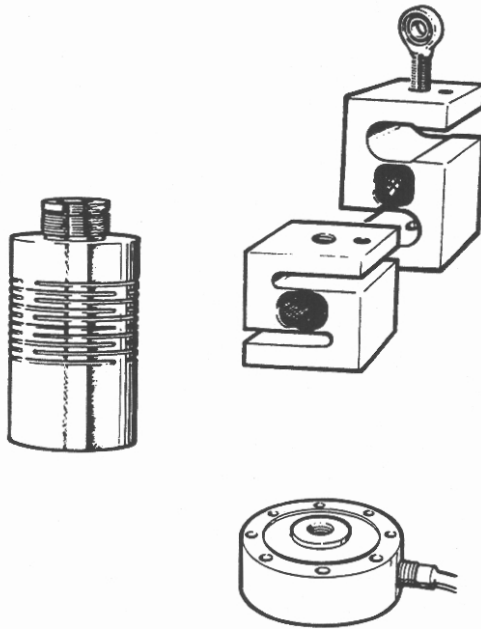


Figure 8-33  
Typical commercial load cells.

## CONSTRUCTION AND OPERATION

Most commercial load cells consist of strain gage devices (usually resistive), an internal bridge circuit, and signal conditioning circuit as indicated by the block diagram in Figure 8-34. As a result, a load cell will produce a temperature compensated voltage output that is linearly proportional to the weight placed on the cell.

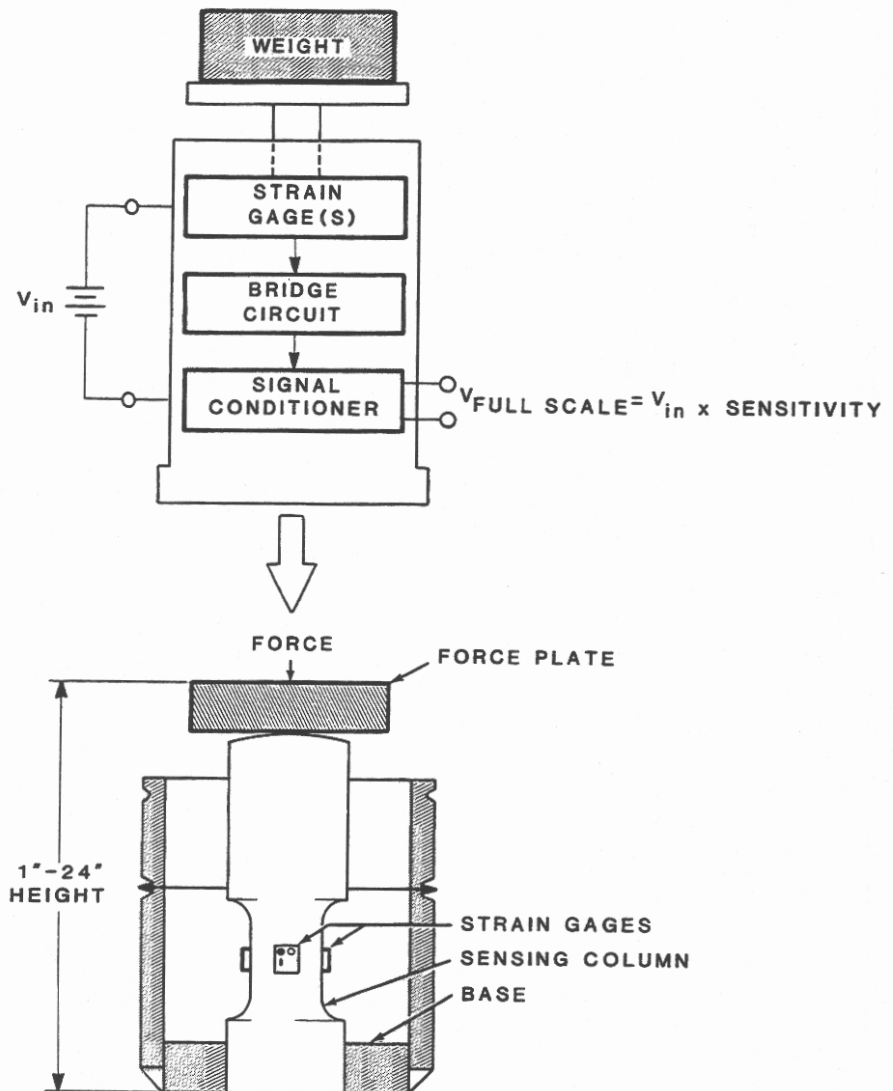


Figure 8-34

A typical strain gage column-type load cell.

Several resistive strain gages are bonded to an internal sensing column as shown, and connected in a Wheatstone bridge configuration. As a force is applied to the top of the column, the column is compressed, resulting in a change in resistance of the bonded strain gages. The resistance change is seen as a change in DC voltage level at the output of the load cell. This type of construction is called a **column load cell**. The height of a column load cell can vary from 1 inch to over 24 inches, depending on the application.

You should also be aware that there are **LVDT load cells** available. As the name implies, these devices use an internal LVDT as shown in Figure 8-35. Here, the LVDT core is attached to a load beam. When a force is applied, it produces a linear deflection of the beam which causes the LVDT core to move within its secondary coils. As you are aware, this produces a change in the secondary voltage amplitude. Note also, that an LVDT load cell must be excited from an AC source. Its output is also an AC signal that must be rectified and filtered, prior to conversion to digital.

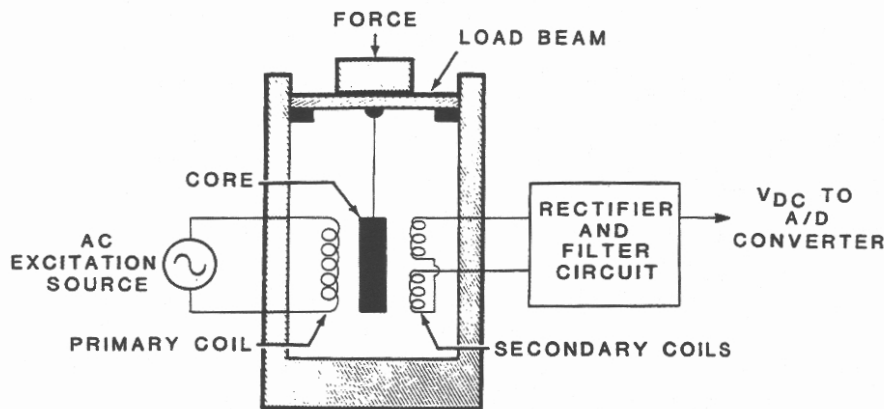


Figure 8-35  
A typical LVDT load cell.

## CHARACTERISTICS

Let's for the time being, confine our discussion to strain gage load cells, since they are typical of the type used in digital control applications. The manufacturer of a strain gage load cell will specify the sensitivity of the cell in millivolts output per volt of excitation, or  $\text{mV/V}$ , at full scale loading. In other words, a  $10 \text{ mV/V}$  load cell operating with a  $12 \text{ V}$  excitation source, will generate  $10 \text{ mV/V} \times 12 \text{ V}$ , or  $120 \text{ mV}$ . This would be the full scale, or maximum output level possible with this excitation source. Smaller loads, or weights, result in smaller output levels.

## APPLICATION CIRCUITS

In most cases, all you must supply to a strain gage load cell is the excitation voltage. The load cell manufacturer will specify a typical excitation voltage as well as a maximum excitation voltage level. The output of the load cell must be amplified to be compatible with the input of an A/D converter for measurement in a digital computer control system.

### Example 8-9:

A given commercial load cell has the following specifications:

Capacity: 0 to 1,000 pounds

Rated Output: 3 mV/V

Excitation Voltage: 10 V nominal, 18 V max.

What output voltage range would you expect from the cell using a 12 V DC source?

### Solution:

The rated output of 3mV/V is specified at a full scale loading of 1,000 pounds. Thus, using a 12 V excitation voltage, the full scale output voltage must be:  $3\text{mV/V} \times 12\text{ V}$ , or 36 mV. The output voltage range would be a linear output from 0 to 36 mV, for weights from 0 to 1,000 pounds, respectively.

## Self-Test Review

11. Name the three types of mechanical stress/strain situations.

\_\_\_\_\_.

\_\_\_\_\_.

\_\_\_\_\_.

12. Explain the difference between stress and strain.

13. Suppose a resistive strain gage changes its resistance from  $350\ \Omega$  to  $348.6\ \Omega$ . What is the strain of the gage?

\_\_\_\_\_.

14. Is the strain in question 13 a tensile or compression strain?

\_\_\_\_\_.

15. Name the two general types of strain gages.

\_\_\_\_\_.

\_\_\_\_\_.

16. What is the gage factor of the strain gage in question 13?

\_\_\_\_\_.

17. Suppose an MPU measures an unstrained output of 5 V from a 12 V bridge circuit and a strained output of 6 V. Also, suppose a semiconductor strain gage with a gage factor of 100 is being used in the bridge. What is the strain of the gage?

\_\_\_\_\_.

18. Explain how to compensate for the effects of lead resistance, temperature, and noise in a strain gage application circuit.



19. List two advantages and two disadvantages of semiconductor strain gages.

Advantages

Disadvantages

\_\_\_\_\_.

\_\_\_\_\_.

\_\_\_\_\_.

\_\_\_\_\_.

20. A commercial load cell has the following specifications:

Capacity: 0 to 500 pounds

Rated Output: 4 mV/V

Excitation Voltage: 12 V nominal,  
30 V max.

What output voltage range would you expect from this cell using a 15 V DC source?

## Answers

11. A. Tensile.  
B. Compression.  
C. Shear.
12. Stress — Is the internal effect on an object caused by an applied force.

Strain — Is the deformation of the object that results from stress.

13. Since the strain gage resistance changed from 350 to 348.6  $\Omega$ , its change in resistance,  $\Delta R$ , is  $-1.4 \Omega$ . Notice that  $\Delta R$  is negative, since the resistance has decreased. Solving for strain in the following equation you get:

$$\Delta R = 2 R_{\text{old}} \times \text{Strain}$$

$$\text{or, Strain} = \frac{\Delta R}{2 R_{\text{old}}}$$

Substituting the given values gives you:

$$\text{Strain} = -1.4 \Omega / 2(350 \Omega)$$

$$= -.002$$

$$= -2000 \times 10^{-6}$$

$$= -2000 \mu$$

14. The strain, in question 13, is a compression strain because it is negative.
15. A. Resistive strain gage (wire and foil bonded).  
B. Semiconductor strain gage.
16. Using the gage factor equation you get:

$$GF = \frac{\Delta R / R_{\text{nominal}}}{\text{Strain}}$$

$$= \frac{-1.4 \Omega / 350 \Omega}{-2000 \mu}$$

$$= 2$$

17. First you must calculate the difference ratio voltage,  $V_r$ , as follows:

$$\begin{aligned} V_r &= \left[ \frac{V_{out}}{V_{in}} \right]_{unstrained} - \left[ \frac{V_{out}}{V_{in}} \right]_{strained} \\ &= \frac{5 \text{ V}}{12 \text{ V}} - \frac{6 \text{ V}}{12 \text{ V}} \\ &= -.083 \text{ V} \end{aligned}$$

Next, calculate the strain as follows:

$$\begin{aligned} \text{Strain} &= \frac{-4V_r}{GF(1 + 2V_r)} \\ &= \frac{-4(-.083 \text{ V})}{100[1 + 2(-.083 \text{ V})]} \\ &= \frac{.332 \text{ V}}{83.4 \text{ V}} \\ &= .003998 \\ &= 3998 \times 10^{-6} \\ &= 3998 \mu \end{aligned}$$

18. First, use at least #18 or #20 shielded twisted cable for the lead wires. Second, use a three wire connection to the gage so that two of the lead wires are placed in adjacent legs of the bridge (See Figure 8-30). Then if the gage is located in an environment at a temperature other than room temperature (25°C), use a dummy gage in an adjacent leg of the bridge circuit (See Figure 8-31).

|     |                   |                          |
|-----|-------------------|--------------------------|
| 19. | <u>Advantages</u> | <u>Disadvantages</u>     |
|     | High sensitivity  | Nonlinear                |
|     | Small size        | Sensitive to temperature |

20. Using a 15 V DC source, the full scale output voltage must be:  $4 \text{ mV/V} \times 15 \text{ V}$ , or 60 mV. Thus, you would expect to see a linear output from 0 V to 60 mV for weight from 0 to 500 pounds.

## UNIT SUMMARY

Position and proximity sensing is extremely important in industrial processes, especially robotics. A position sensor measures the actual position of an object, while a proximity sensor detects and/or measures the nearness of objects. Position and proximity sensors include potentiometric, capacitive, inductive, magnetic, and optical sensors. Each type has its own unique characteristics that must be considered for a given application.

Force sensing involves the measurement of mechanical forces exerted on an object. The internal effect on the object caused by the force is called stress, while the resulting object deformation is called strain. The forces acting on an object are measured by measuring the amount of resulting deformation, or strain of the object using strain gages.

A strain gage attached to an object changes its resistance in proportion to an applied force. There are wire bonded and foil bonded resistive strain gages, as well as semiconductor strain gages. In general, the strain gage is used as part of a DC Wheatstone bridge circuit, so that small resistance changes can be detected and measured. A load cell is a commercial device that uses strain gages to measure weight, from several pounds to several million pounds.



*Unit 9*

**CONTROL DEVICES AND CIRCUITS**

## CONTENTS

|                                                      |      |
|------------------------------------------------------|------|
| Introduction .....                                   | 9-3  |
| Unit Objectives .....                                | 9-4  |
| MPU Control of DC and AC Loads .....                 | 9-5  |
| Open Collector Drivers .....                         | 9-6  |
| Bipolar Power Transistors .....                      | 9-8  |
| Bipolar Transistor Arrays .....                      | 9-9  |
| Power MOSFETs .....                                  | 9-11 |
| Peripheral Power Drivers .....                       | 9-12 |
| Optocouplers .....                                   | 9-13 |
| Optoisolator Characteristics .....                   | 9-16 |
| Thyristors .....                                     | 9-16 |
| SCRs .....                                           | 9-17 |
| TRIACs .....                                         | 9-19 |
| SCR and TRIAC Control Circuits .....                 | 9-21 |
| Relay and Solenoid Actuators .....                   | 9-26 |
| Solid-State Relays .....                             | 9-27 |
| Electromechanical Relays .....                       | 9-30 |
| Solenoids .....                                      | 9-34 |
| Self-Test Review .....                               | 9-35 |
| Answers .....                                        | 9-37 |
| MPU Control of DC Motors .....                       | 9-38 |
| Controlling Permanent Magnet DC Motors .....         | 9-38 |
| Using a D/A Converter for Motor Control .....        | 9-38 |
| Using Pulse-Width Modulation for Motor Control ..... | 9-39 |
| Controlling Wound-Field DC Motors .....              | 9-44 |
| Stepper Motors .....                                 | 9-47 |
| Bipolar Permanent Magnet Stepper .....               | 9-48 |
| Bifilar, or Unipolar, Motor .....                    | 9-50 |
| Controlling Stepper Motors .....                     | 9-52 |
| Bipolar Control .....                                | 9-53 |
| Unipolar Control .....                               | 9-54 |
| The MPU Interface .....                              | 9-59 |
| Self-Test Review .....                               | 9-62 |
| Answers .....                                        | 9-63 |
| Unit Summary .....                                   | 9-64 |

## ***Unit 9***

# **CONTROL DEVICES AND CIRCUITS**

## **INTRODUCTION**

Recall that most commercial and industrial microprocessor applications require a closed loop control system that consists of three important parts: sensing, decision making, and control. Up to now, your study of microprocessor applications has concentrated on the sensing and decision-making portions of the process control loop, except for the discussion of DACs in Unit 6.

After the microprocessor makes a decision based on a sensed condition, it must output a control message to a final control element, or load, such as a valve, relay, motor, or solenoid. In most cases, the digital signals from the microprocessor cannot be used directly to control the final control element. In this unit, you will be acquainted with devices that are used by microprocessor circuits to control relatively high power loads. These control devices include peripheral power drivers, transistor arrays, SCRs, TRIACs, and solid-state relays.

In addition, many microprocessor applications require that the microprocessor control a motor. DC motors are preferred over AC motors since they are much more easy to control with a microprocessor circuit. For this reason, we have included sections on DC motors and stepper motors in this unit. A stepper motor is a special type of DC motor which is used extensively in robotics.



## UNIT OBJECTIVES

1. State several microprocessor control applications for simple control devices.
2. Design a microprocessor-controlled SCR or TRIAC circuit.
3. Explain how a microprocessor can control the effective current to a load using an SCR or TRIAC.
4. State the advantages of using an opto-isolator in a microprocessor control circuit.
5. List at least three considerations that must be taken into account when you are using solid-state relays to control high current loads.
6. Explain how to control the speed of the various types of DC motors.
7. Describe the operation of bipolar and unipolar stepper motors.
8. Explain how to control the direction of rotation, amount of rotation, and speed of a stepper motor.
9. Design and explain a microprocessor/stepper motor interface and control circuit.

## MPU CONTROL OF DC AND AC LOADS

The D/A converter discussed in Unit 6 is typically used when the final control element requires a range of control signals, from some minimum to some maximum value. You found that the speed and direction of some DC motors are controlled in this manner. A bipolar DAC output is used to control the motor's direction and speed by applying different voltage levels to the motor. The DAC output is one polarity for rotation in one direction and the opposite polarity for rotation in the opposite direction. The speed is controlled by the voltage level — as the voltage output of the DAC increases in one direction, so does the speed of the motor. In summary, you could say that this type of control requires a variable control signal that can take-on a range of values from a given minimum to a given maximum value.

It turns out that there are numerous control applications that do not require a variable control signal. These are on/off operations that use on/off control elements such as relays, solenoids, valves, and even motors. A valve can be either open or closed, a motor might be turned on or off and run at a constant speed. This type of control does not require a D/A converter. Instead, a single digital pulse is generated by the MPU and applied to an electronic control device, such as a power transistor, that is used to activate the final control element. The digital signal cannot be used directly, since it does not have enough power to drive the final control element. This control application is illustrated in Figure 9-1.

In this section, you will learn about several on/off electronic control devices and circuits that are used to control DC and AC loads. Then, in the next two sections, you will see how these devices are used to control components such as relays, solenoids, valves, and motors.

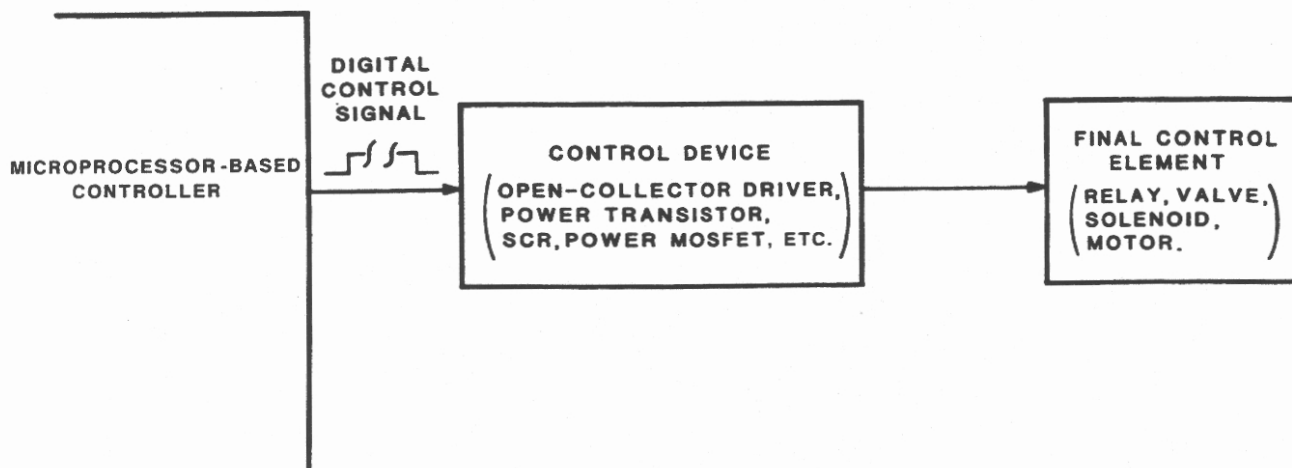


Figure 9-1

An electronic control device is often required to control a high power final control element.

## Open Collector Drivers

Open collector drivers are digital devices that act like a switch in a pull-up resistor circuit. Look at Figure 9-2. When the switch is open, there is an infinite resistance path from ground and the circuit output is pulled high to the +V pull-up resistor potential. However, when the switch is closed, there is a low resistance path from ground through the open collector device. As a result, the circuit output is at ground potential, or 0 volts. Stated another way, when the switch is closed, current is sunk to ground through the open collector device. The "switch" in the device is actually an integrated transistor. Consequently, the amount of current that can be sunk is limited by the internal current-carrying capabilities of the internal transistors.

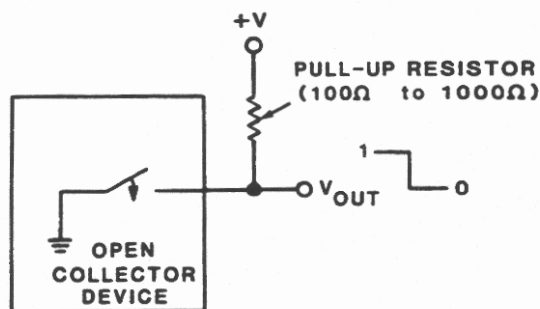


Figure 9-2

An open collector device acts like a switch in a pull-up resistor circuit.

In summary, you could say that an open collector device makes or breaks a circuit path to ground. This feature allows open collector devices to drive low voltage and low current loads such as incandescent lamps, LEDs, and small reed relays. When such a device is connected in series with an open collector pull-up resistor as illustrated in Figure 9-3, a logic 0 output creates a current flow from ground, thereby activating the device. The pull-up resistor is used to limit the amount of current in the circuit. Its size is not critical, but usually ranges between 100 ohms and 1000 ohms, depending on the pull-up voltage potential.

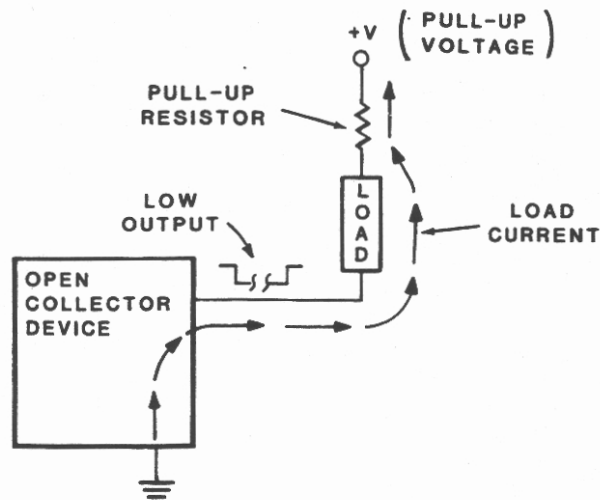


Figure 9-3

A load is connected in series with the open collector output and pull-up voltage.

Open collector drivers are rated by their maximum current-sinking capabilities when the output is low, and the maximum pull-up potential when the output is high. Several popular 7400-series open collector drivers are listed in Table 9-1, along with their maximum ratings.

| <u>Open Collector Device</u> | <u>Maximum Sink Current</u> | <u>Maximum Pull-Up Voltage</u> |
|------------------------------|-----------------------------|--------------------------------|
| 7405 Hex Inverter            | 12 mA                       | +5 V                           |
| 7406 Hex Inverter            | 40 mA                       | +30 V                          |
| 7407 Quad 2-bit NAND         | 16 mA                       | +15 V                          |
| 7433 Quad 2-bit NOR          | 48 mA                       | +5 V                           |

Table 9-1

Several common 7400 series open collector devices.

## Bipolar Power Transistors

Power transistors are required when the final control element requires higher current and voltage levels than can be supplied by open collector devices. The use of power transistors as on/off control devices is illustrated by Figure 9-4. In Figure 9-4A, an NPN transistor is connected in series with the final control element, or load. With a TTL logic 0, or ground, applied to the base, the transistor acts like an open switch and conducts very little current. However, when the controller applies a logic 1, or 5 V, TTL level to the base, the transistor acts like a closed switch and conducts electron current from ground, thereby activating the control element. A PNP power transistor control circuit is shown in Figure 9-4B. Here a logic zero from the controller activates the load, while a logic 1 opens the transistor switch and deactivates the load. That's all there is to it!

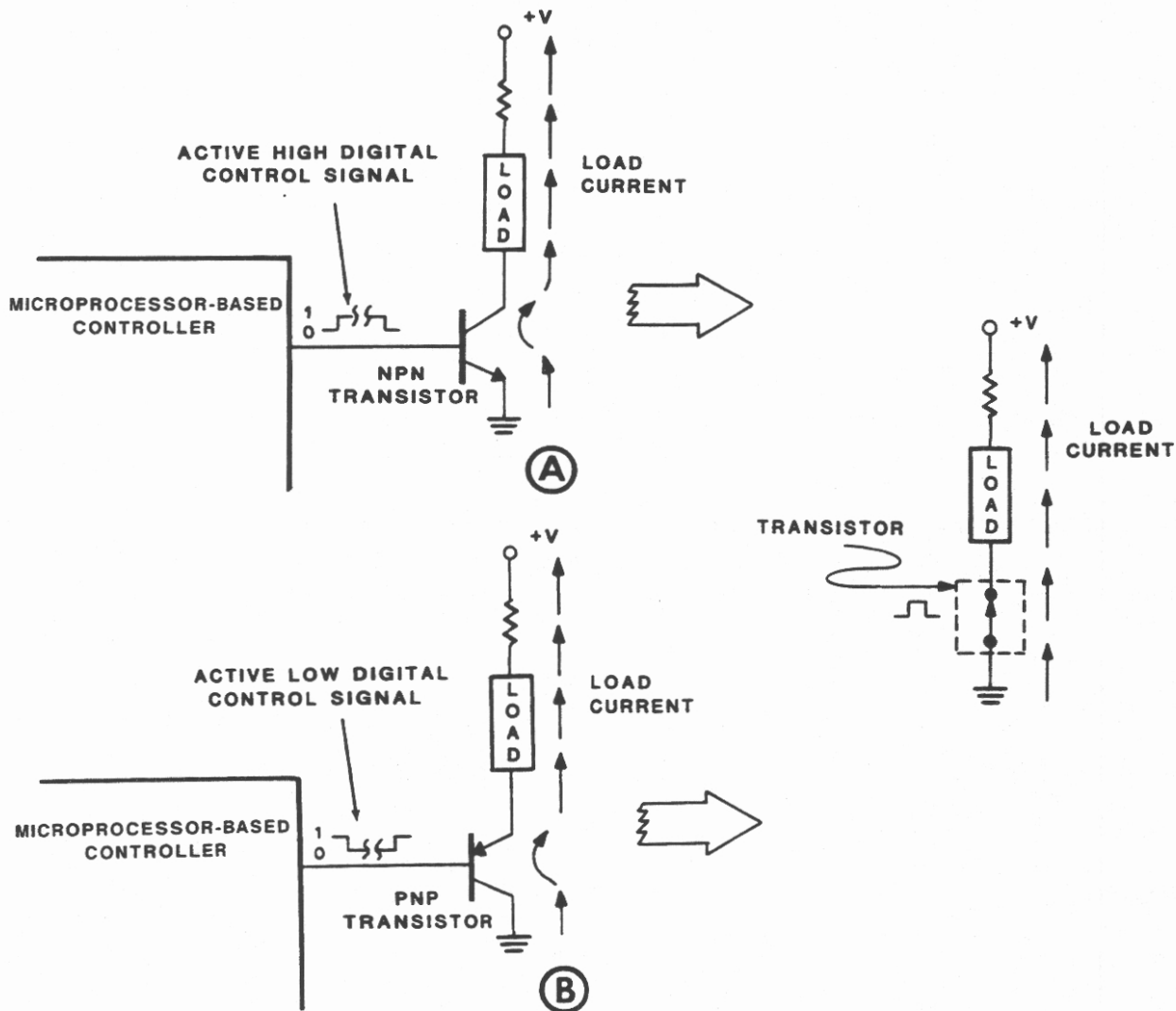


Figure 9-4

- A. An NPN control circuit.
- B. A PNP control circuit.
- Both act as a switch in series with the load.

Bipolar power transistors are commercially available with current ratings that range from .5 to 300 amps and voltage ratings from 25 to 1500 volts. In addition, they are relatively fast switching, which means that they can easily "follow" a digital control signal. Your application will determine the type of transistor best suited for the job. You do not need an expensive 100 amp power transistor for a control element that only requires 200 milliamps. In this case, a general purpose small signal transistor will do the job much more economically.

## BIPOLAR TRANSISTOR ARRAYS

Bipolar transistors are also available in IC packages called **transistor arrays**. A transistor array is simply an IC that contains several integrated driver transistors. The inputs to the array are usually TTL or CMOS compatible so that they can be driven by digital control signals. The individual transistors are wired into the control circuit as if they were separate discrete devices. Their outputs can be used to control relatively high current and high voltage loads.

A big advantage of using transistor arrays over discrete transistors in a control circuit is that the transistors within an array have very close electrical and thermal properties, since they are integrated into the same chip. One disadvantage is that they cannot handle as much power as discrete power transistors. The transistors within most transistor arrays are limited to less than 1 watt of power dissipation, while discrete power transistors are available that can dissipate several hundred watts.

Several transistor arrays are shown in Figure 9-5. The simple array in Figure 9-5A contains three NPN transistors, while the one in Figure 9-5B contains three NPN and two PNP transistors. Some arrays, like the one in Figure 9-5C, include Darlington connections, diodes, and silicon controlled rectifiers (SCRs) that can be used for a variety of control applications. For more application information, you must consult the manufacturers of these devices. Manufacturers of transistor arrays include RCA, National Semiconductor, and Sprague, among others.

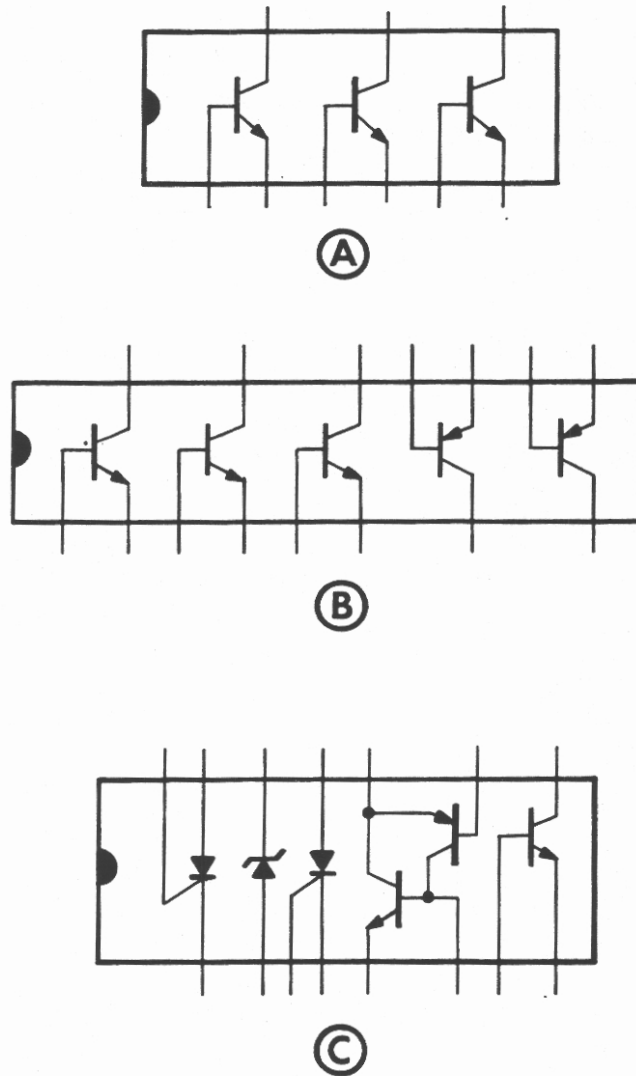


Figure 9-5  
Several typical transistor array ICs.

## Power MOSFETs

The abbreviation MOSFET stands for Metal Oxide Semiconductor Field Effect Transistor. These devices can be used as electronic switches much like bipolar power transistors. As you can see in Figure 9-6, the MOSFET acts like an open switch when the control signal is low. The MOSFET switch closes and allows current to flow through the load when the control signal goes high.

MOSFETs have the advantage of a very high input resistance, and thus do not tend to load the digital control signal. In fact, several MOSFET devices can be controlled by the same digital control signal. However, they cannot switch as fast and have lower current ratings than bipolar power transistors. Although most power MOSFETs available today cannot handle more than 15 amperes, technological improvements are changing this limitation. Future power MOSFET devices are sure to have both the speed and current carrying capabilities of their bipolar counterparts.

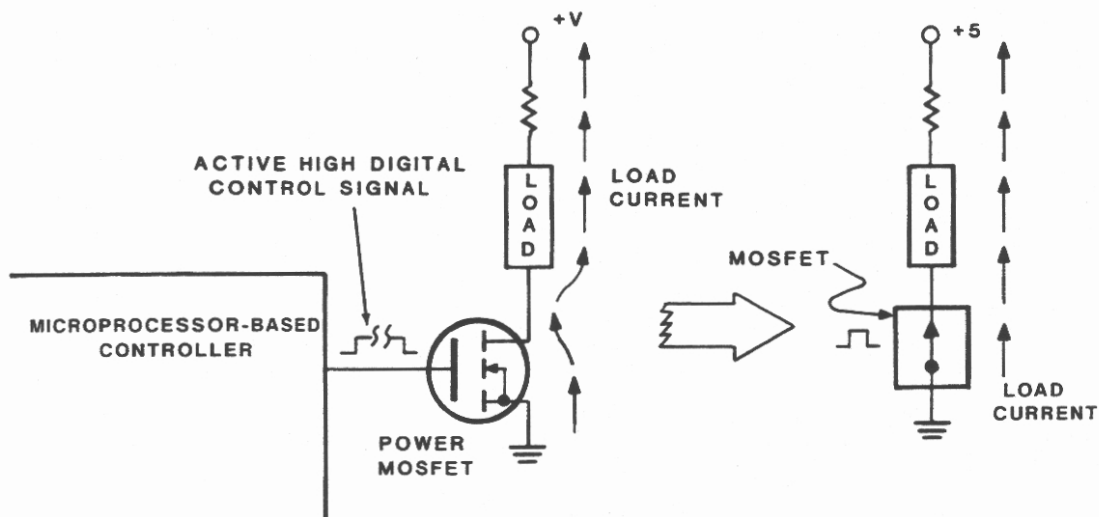


Figure 9-6  
A power MOSFET control circuit.



## Peripheral Power Drivers

A **peripheral power driver** is a device that includes both digital logic gates and driver transistors in the same IC package. One such device is shown in Figure 9-7. Observe that this IC has two separate driver circuits, each consisting of a 2-bit AND gate input and a power transistor output. The gate inputs are TTL and/or CMOS compatible, and the driver transistors provide open collector outputs. Depending on the device, peripheral driver outputs can sink up to 1 amp and be used with a pull-up voltage of up to 50 V. Several popular peripheral power driver ICs are listed in Table 9-2, along with their current sinking and pull-up voltage ratings. Notice that these drivers use different logic functions such as AND, OR, NAND, and NOR.

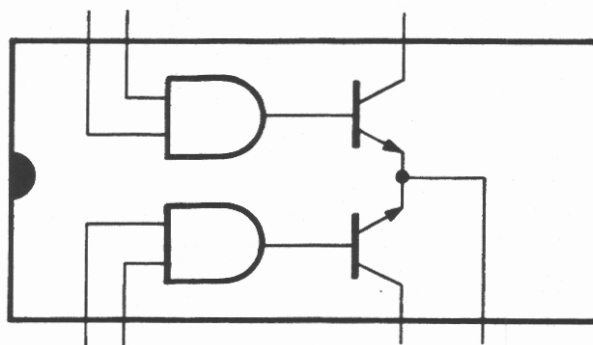


Figure 9-7  
A typical peripheral power driver.

| Device  | Drivers/<br>Package | Input<br>Compatibility<br>(Logic) | Logic<br>Function | Max.<br>Sink<br>Current<br>(mA) | Max.<br>Pull-Up<br>Voltage<br>(volts) |
|---------|---------------------|-----------------------------------|-------------------|---------------------------------|---------------------------------------|
| DS75450 | 2                   | TTL                               | AND               | 300                             | 20                                    |
| DS75451 | 2                   | TTL                               | AND               | 300                             | 20                                    |
| DS75452 | 2                   | TTL                               | NAND              | 300                             | 20                                    |
| DS75453 | 2                   | TTL                               | OR                | 300                             | 20                                    |
| DS75454 | 2                   | TTL                               | NOR               | 300                             | 20                                    |
| DS75460 | 2                   | TTL                               | AND               | 300                             | 30                                    |
| DS75461 | 2                   | TTL                               | AND               | 300                             | 30                                    |
| DS75462 | 2                   | TTL                               | NAND              | 300                             | 30                                    |
| DS75463 | 2                   | TTL                               | OR                | 300                             | 30                                    |
| DS75464 | 2                   | TTL                               | NOR               | 300                             | 30                                    |
| DS3631  | 2                   | CMOS                              | AND               | 300                             | 40                                    |
| DS3632  | 2                   | CMOS                              | NAND              | 300                             | 40                                    |
| DS3633  | 2                   | CMOS                              | OR                | 300                             | 40                                    |
| DS3634  | 2                   | CMOS                              | NOR               | 300                             | 40                                    |
| DS3611  | 2                   | TTL/CMOS                          | AND               | 300                             | 50                                    |
| DS3612  | 2                   | TTL/CMOS                          | NAND              | 300                             | 50                                    |
| DS3613  | 2                   | TTL/CMOS                          | OR                | 300                             | 50                                    |
| DS3614  | 2                   | TTL/CMOS                          | NOR               | 300                             | 50                                    |

Table 9-2

Several Commercially Available Peripheral Power Driver ICs.

## Optocouplers

Optocouplers, often called **optoisolators**, can be classified as control devices, since they are used by a digital system to control external events. They are extremely important in protecting the digital computer circuits and their human operators from high power load circuits. Most optoisolators can provide several kilovolts of isolation between the digital controller and load circuits. This means that it would take several kilovolts to break down the optical isolation provided by these devices and damage the microprocessor controller or its operator.

Optoisolators operate using an infrared LED, called an IRED, and a phototransistor as shown in Figure 9-8A. Compare this to the schematic representation shown in Figure 9-8B. Notice that the IRED and phototransistor are separated by a layer of optical-quality glass and are coupled only by the infrared light beam produced by the IRED. The amount of electrical current applied to the IRED terminals determines the intensity of the IRED. This, in turn, controls the conductivity of the phototransistor.

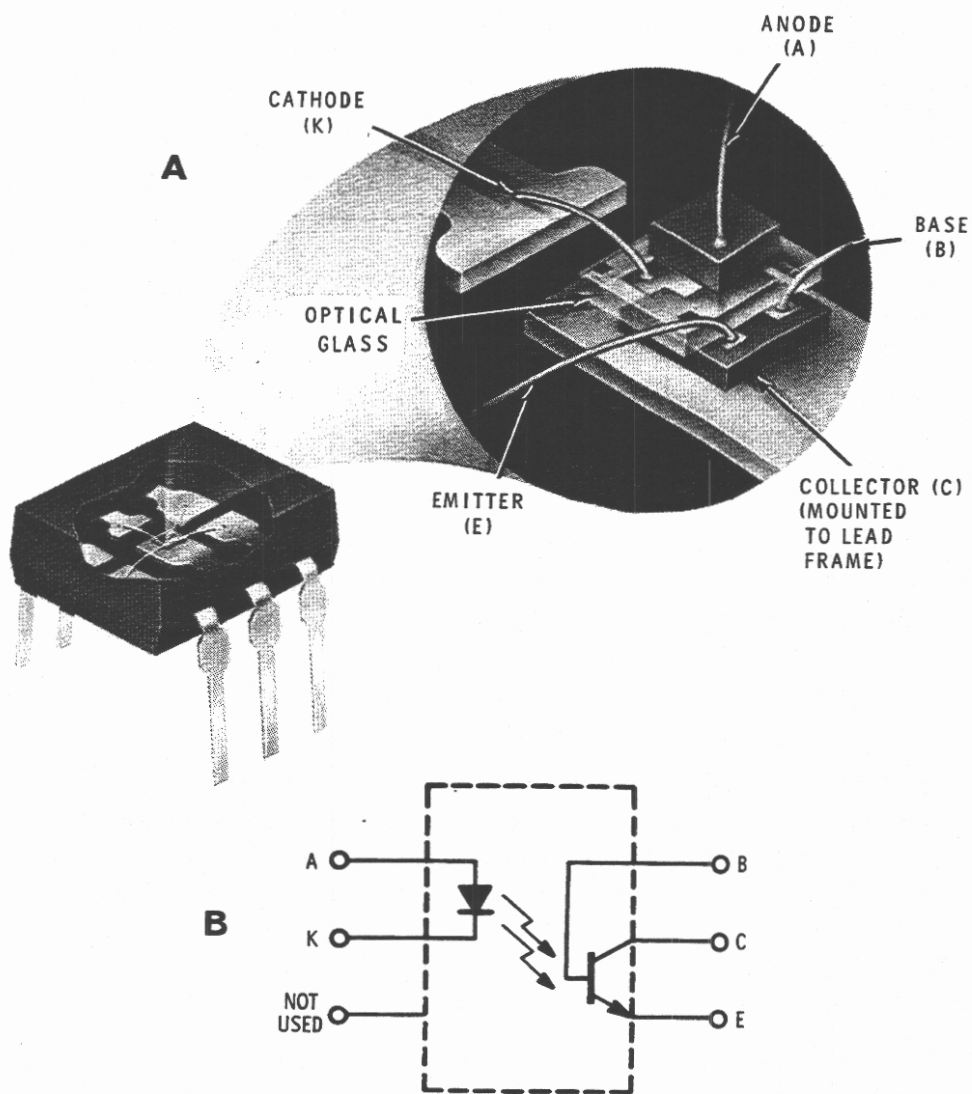


Figure 9-8

A typical optocoupler, or optoisolator, contains an IRED and a phototransistor.

Since the input diode and output transistor are completely separated, they need not use the same power source or ground return paths. This makes it possible to greatly reduce or eliminate electromagnetic interference, leakage currents, dangerous ground loop currents, and other undesirable features associated with standard electrical and electromagnetic signal coupling.

The most important feature of an optoisolator is the electrical isolation that it provides between the microprocessor control circuit and a higher power analog load circuit. From Figure 9-9 you can see that the optoisolator *is* the interface between the digital and load circuits. Here, a logic 1 generated by the digital circuit turns-on the IRED, which causes the phototransistor to conduct and supply current to the load. Conversely, a logic 0 turns-off the IRED, the phototransistor acts like an open switch and no current is passed through the load. Thus, the optoisolator is acting like a digitally controlled switch for the load.

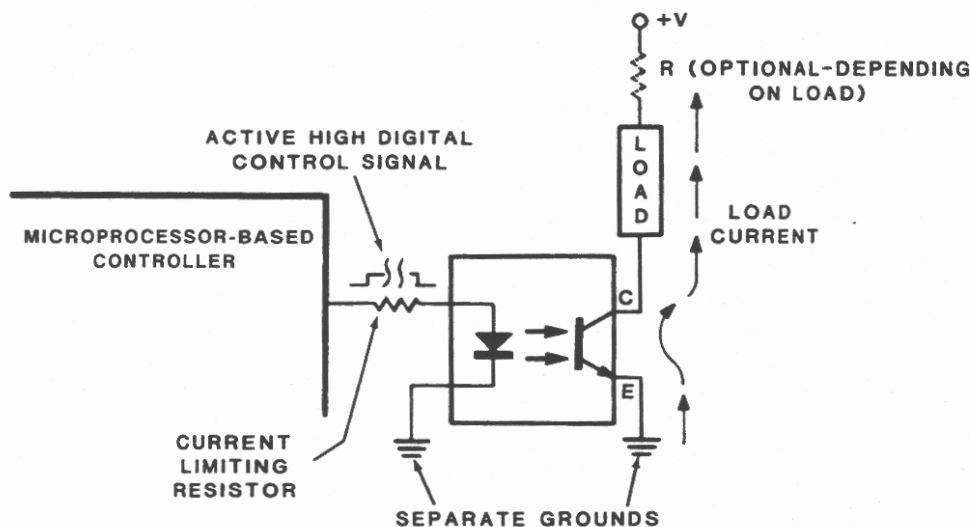


Figure 9-9

An optoisolator provides isolation between the digital and load circuits.

Observe that only the collector and emitter leads of the phototransistor are used. The base lead is not required and is often "clipped-off" for this application. In addition, the input and output lines of the optoisolator can operate from two separate grounds. Isolated grounds will prevent lethal shock-hazard situations. Furthermore, the microprocessor circuit and its operator are protected against the higher voltages of the load circuit, since there is no direct connection between the two circuits.

Optical isolation is especially useful in biomedical equipment, where optocouplers are used to provide electrical isolation between the patient and potentially dangerous voltages within a medical instrument. Also, since the diode and the phototransistor cannot reverse their roles, the light path through the optoisolator is strictly one way. This makes it possible to control industrial machine circuits with computer logic circuits without switching transients, (glitches) and power surges being fed back into the sensitive computer circuits.

## OPTOISOLATOR CHARACTERISTICS

Isolation resistance refers to the DC resistance between the input and the output of the device. In an optoisolator, this is a very high resistance, typically 100 to 1000 gigaohms ( $10^{11}$  to  $10^{12}$  ohms).

The isolation voltage rating of an optoisolator refers to the maximum voltage difference that can be safely applied between the input and output of the device without danger of insulation breakdown. The rating depends on the type of insulation material, method of construction, and rate at which the voltage is applied. Since the rate of voltage change is a factor, this rating is usually stated in several ways. For instance, a typical data sheet might list the isolation voltage rating as follows:

Surge Isolation Voltage (Input to Output)

1500 volts (peak), 1060 volts (rms)

Steady-State Isolation Voltage (Input to Output)

950 volts (peak), 660 volts (rms)

It should be mentioned that these devices are not self-healing. If the isolation voltage rating is exceeded to the point that insulation breakdown occurs, the device is permanently damaged. At the very least, a carbonized path is formed as a result of high currents passing through the insulation material. In extreme cases, lead wires will melt, resulting in a possible short circuit between input and output connections. Therefore, when you use an optocoupler to isolate microprocessor control circuits, you should select a device with a sufficiently high voltage rating. Under extreme conditions, additional fuse protection may be necessary.

## Thyristors

The term thyristor is a general term for a broad range of semiconductor devices which are used as electronic switches. Thyristors are generally used in heating/cooling environmental control systems. An example of their application is in light control and industrial control, where DC and AC power must be controlled.

Thyristor devices that are most common in microprocessor-based control applications are the **silicon controlled rectifier**, or SCR, and the **bidirectional triode thyristor**, or TRIAC. In general, the SCR is used to control DC loads and the TRIAC is used to control AC loads. A brief review of these devices is included in this section. Then you will learn how to use SCRs and TRIACs to control high power DC and AC loads using a digital computer signal.

## SCRs

The SCR is basically a rectifier. However, unlike a standard diode rectifier, the SCR can be turned on and off, thereby providing a switching action that can be used to control current. Actually, we could also call an SCR a solid-state DC relay.

As you can see from the schematic symbol in Figure 9-10A, the SCR has three external leads: an anode, cathode, and gate lead. In normal operation, the SCR anode/cathode junction is forward-biased as shown in Figure 9-10B. However, simply forward-biasing the SCR does not permit it to conduct. To turn the SCR on, you must apply a suitable gate current. Notice from Figure 9-10B that a switch (S) is used to apply or remove a gate current that is developed from a DC voltage source and a current limiting resistor,  $R_g$ . When switch S is closed, gate current is applied and the SCR conducts electron current from its cathode to anode leads. The gate current only needs to be applied momentarily to turn on, or *trigger*, the SCR. Once triggered, the SCR will continue to conduct until the anode/cathode junction is no longer forward-biased. Simply removing the gate current *does not* turn-off the SCR.

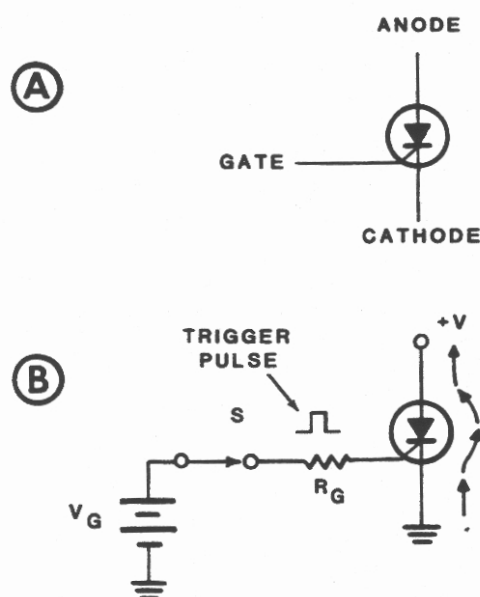


Figure 9-10

- A. A schematic symbol of an SCR.
- B. A properly biased SCR.

The fundamental idea of using an SCR in control circuits is that a very small gate current can control a very large cathode-to-anode load current. SCRs are primarily used to rectify and control current from an AC source and apply it to a DC load as illustrated in Figure 9-11. When the SCR is triggered by the application of a suitable gate voltage ( $V_G$ ), the SCR will conduct for the positive half-cycle of the input voltage waveform. Then it will turn-off during the negative half-cycle, since it is reverse-biased during this half-cycle. The SCR will remain off unless another gate pulse is applied during the next positive half-cycle. If the gate pulse is applied at the beginning of each positive half-cycle, the current through the load will be a half waveform, as shown in Figure 9-11B.

You might be wondering why a standard power rectifier isn't used rather than an SCR in this circuit, since a standard rectifier would produce the same load current. The reason is that the timing of the gate pulse in an SCR can be shifted as shown in Figure 9-11C. This results in reducing the average current to the load. Changing the time at which the gate pulse is applied during the positive half cycle changes the average, or equivalent, DC current applied to the load. The outcome of this change is that more or less current is applied to the load, depending on the timing of the gate pulse.

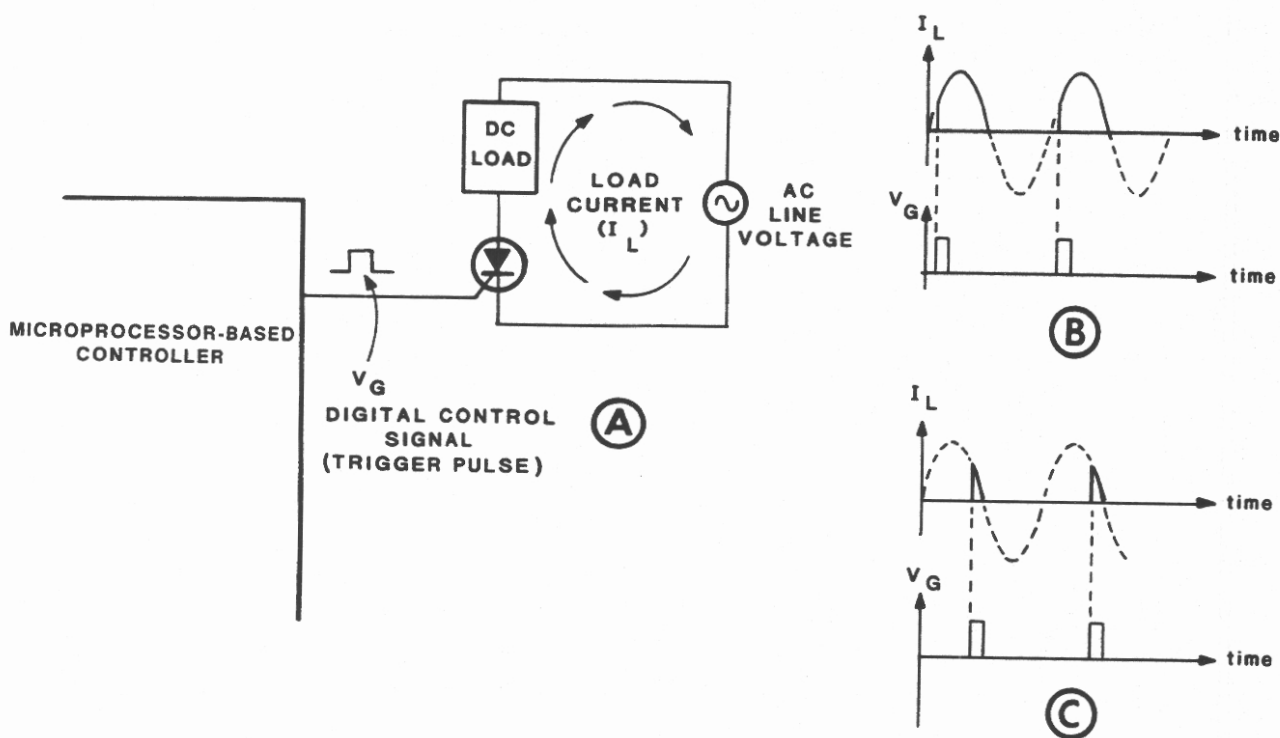


Figure 9-11

- A. Using an SCR to control the current through a DC load.  
 B. & C. The firing angle determines the amount of current flow through the load.

The point at which the SCR is triggered on the AC source waveform is called the **firing angle**. The term "angle" refers to the angular displacement of the sine wave. Thus, if an SCR were triggered at the peak of the positive half-cycle, the firing angle would be  $90^\circ$ .

Now, assume that the load is a resistive heating element, incandescent light filament, or DC motor. By controlling the gate pulse timing, you can control the temperature of the heating element, the intensity of the light, or the speed of the DC motor.

## TRIACs

A TRIAC is basically two SCRs which are in parallel but are connected in opposite directions. This idea is illustrated by the schematic symbol of a TRIAC shown in Figure 9-12. The major difference between the TRIAC and SCR is that the TRIAC conducts current in both directions, while the SCR only conducts current in one direction. For this reason, the TRIAC is ideal for controlling power to AC loads, as illustrated in Figure 9-13.

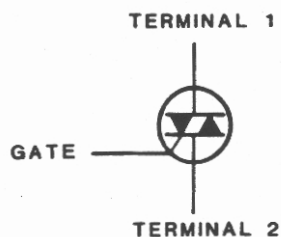


Figure 9-12  
Schematic symbol for a TRIAC.



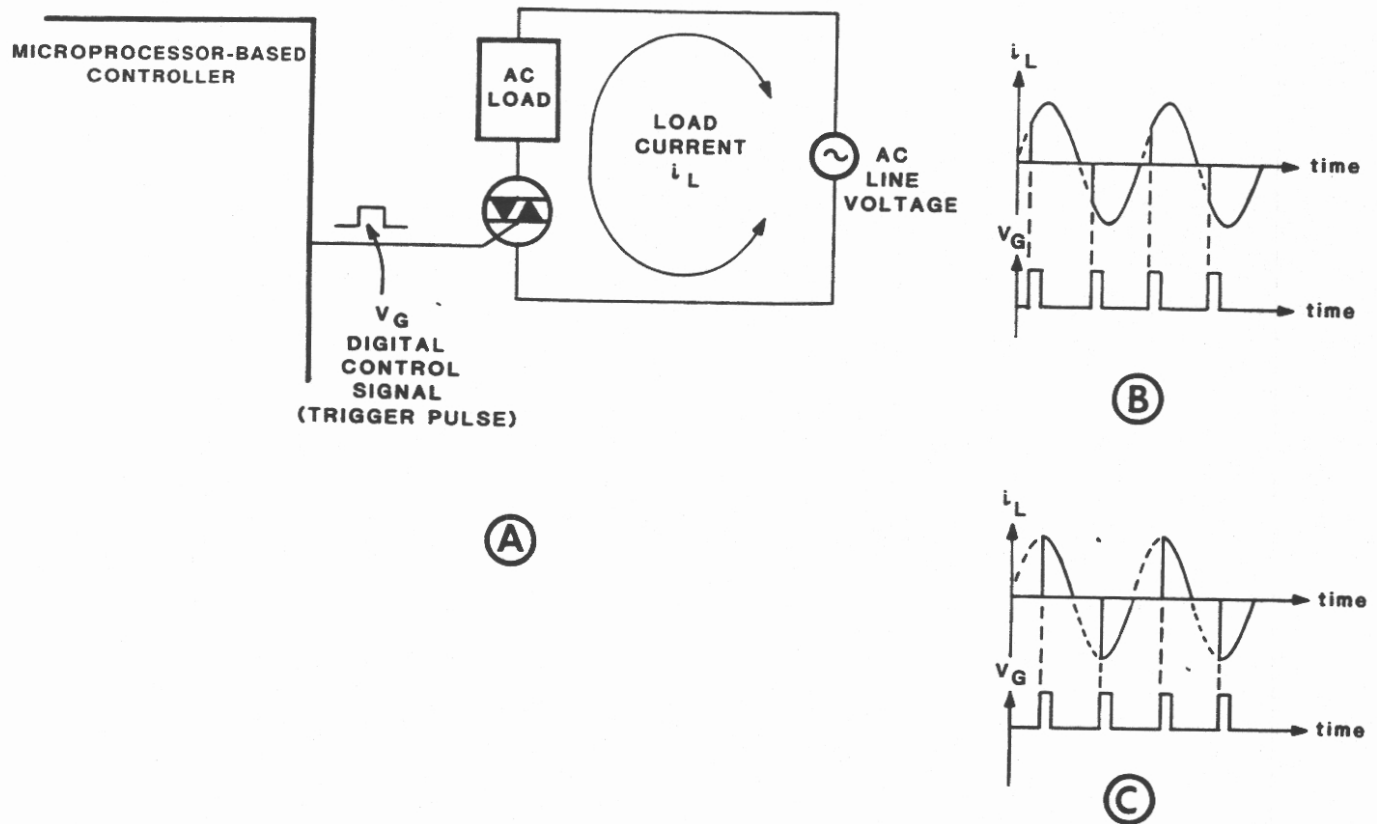


Figure 9-13  
Using a TRIAC to control an AC load.

Like the SCR, the TRIAC is turned on, or triggered, by a triggering pulse that is applied to its gate lead. The trigger pulse can be positive or negative; however, it must be applied on each successive half-cycle of the source waveform to produce a bidirectional AC current flow through the load. The trigger pulse turns the TRIAC on for a given half-cycle. The TRIAC remains on until the AC voltage reverses its direction in the next half-cycle and turns the TRIAC off. Application of another trigger pulse on the next half-cycle turns the TRIAC on again and allows it to conduct for the remainder of the AC cycle. Changing the trigger pulse timing, or firing angle, changes the effective current applied to the load, as illustrated by the waveforms in Figure 9-13B and C. The AC load can be a resistive load such as in electric heating and lighting systems, or a reactive load as in AC motors.

In general, TRIACs have lower current ratings than SCRs and cannot compete with SCRs in applications where extremely large currents must be controlled. TRIACs are available that can handle rms currents as high as 25 amperes. By comparison, SCRs can be readily obtained with average current ratings as high as 700 or 800 amperes, and some are rated even higher. In addition, TRIACs are designed for low frequency (50 to 400 Hz) applications, while SCRs can be used at frequencies up to 30 kHz. Therefore, in certain applications where full control of the AC signal is required, two SCRs in parallel but connected in opposite directions may operate more efficiently than a TRIAC; while in other applications the exact opposite is true.

## SCR AND TRIAC CONTROL CIRCUITS

Now, you are probably wondering how a microprocessor-based controller can be used to control an SCR or TRIAC. To control an SCR or TRIAC, a microprocessor must control the timing, or firing angle, of the gate lead trigger pulse. To do this, the microprocessor must perform three tasks:

1. Determine the zero-crossing point of the AC source waveform.
2. Execute a software delay to establish the firing angle of the SCR or TRIAC.
3. Generate the triggering pulse.

The circuit shown in Figure 9-14 can be used by the microprocessor-based controller to determine the zero-crossing point of the AC source waveform. This zero-crossing detector circuit uses a transformer to step-down the AC line voltage and apply it to the input of a zero-crossing detector op amp circuit.

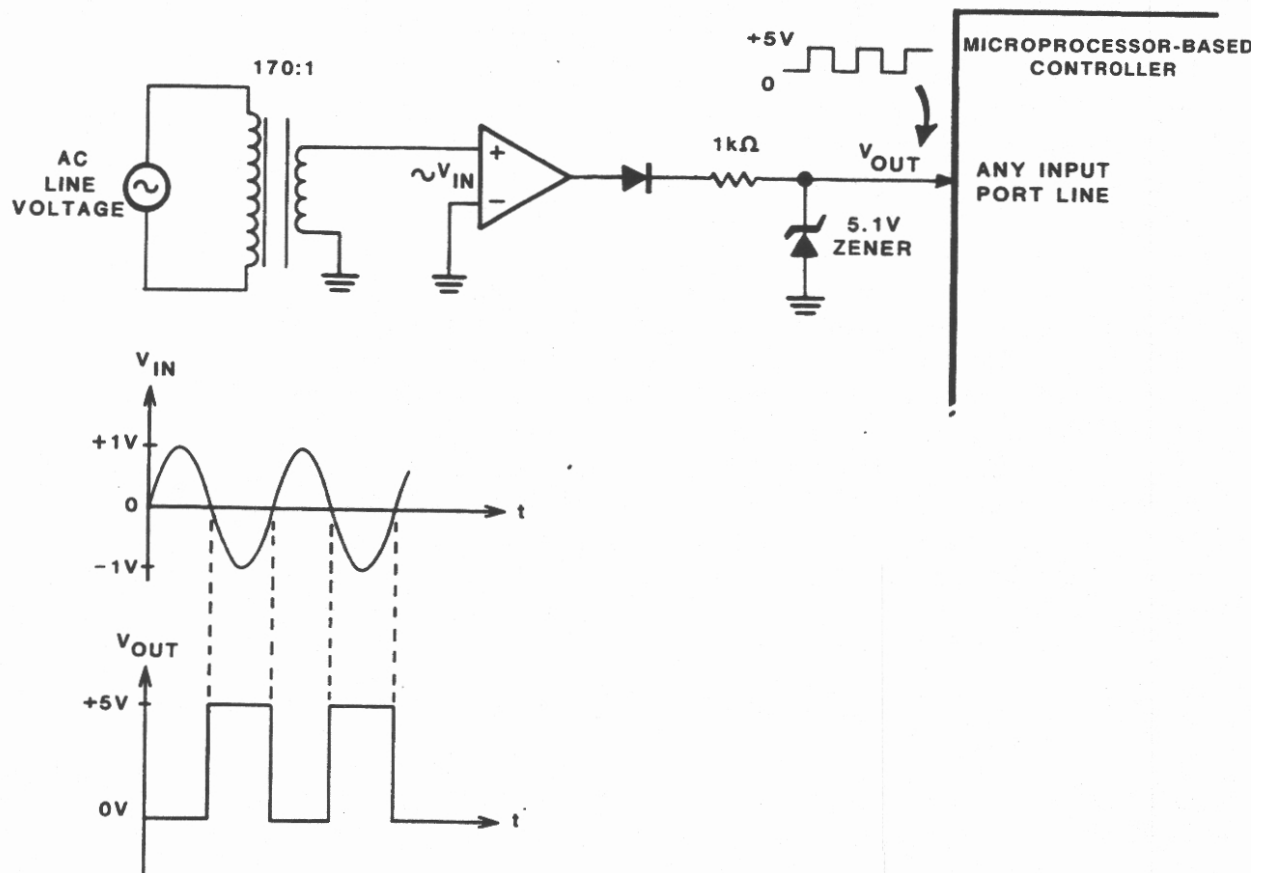


Figure 9-14

A zero-crossing detector using an op amp comparator circuit.

Each time the AC input crosses zero, the output of the zero-crossing detector makes a transition from 0 V to +5 V or from +5 V to 0 V as shown by the waveforms in Figure 9-14. Since this output is TTL compatible, it can be applied directly to an input port line of the controller as shown. The controller must be programmed to periodically check the logic bit status of the input port line. A logic transition will occur each time the AC voltage crosses zero. With a 60 Hz AC line signal, zero-crossing will occur every 8.33 milliseconds. Thus, the bit status polling loop must be executed at least once every millisecond to "catch" the zero-crossing point.

After the zero-crossing point is detected, the MPU must execute a software time delay, and then generate the triggering pulse. The amount of time delay between zero-crossing and pulse generation determines the firing angle of the SCR or TRIAC. The timing waveforms for an SCR are shown in Figure 9-15, and those associated with a TRIAC are shown in Figure 9-16. Notice from the timing diagrams that the software delay is actually equal to the amount of time during the half-cycle that the SCR or TRIAC is to be *turned-off*, or not conducting, during a given half-cycle. Study these diagrams to get the idea. The difference between TRIAC and SCR timing is that for SCR control, the controller only needs to detect a high-to-low transition from the zero-crossing detector circuit. A high-to-low transition indicates the beginning of the positive half-cycle of the input waveform. Once detected, the MPU executes a software time delay to adjust the firing angle, and then generates the SCR trigger pulse.

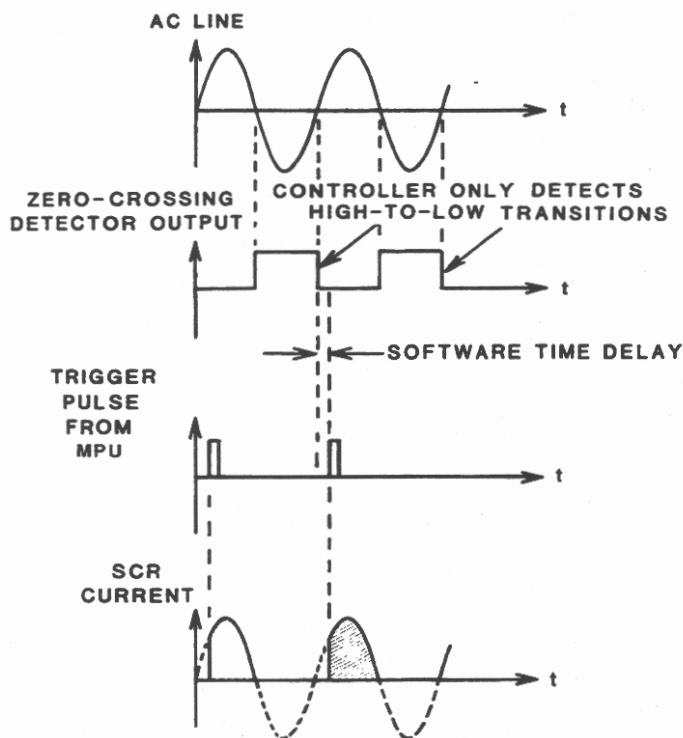
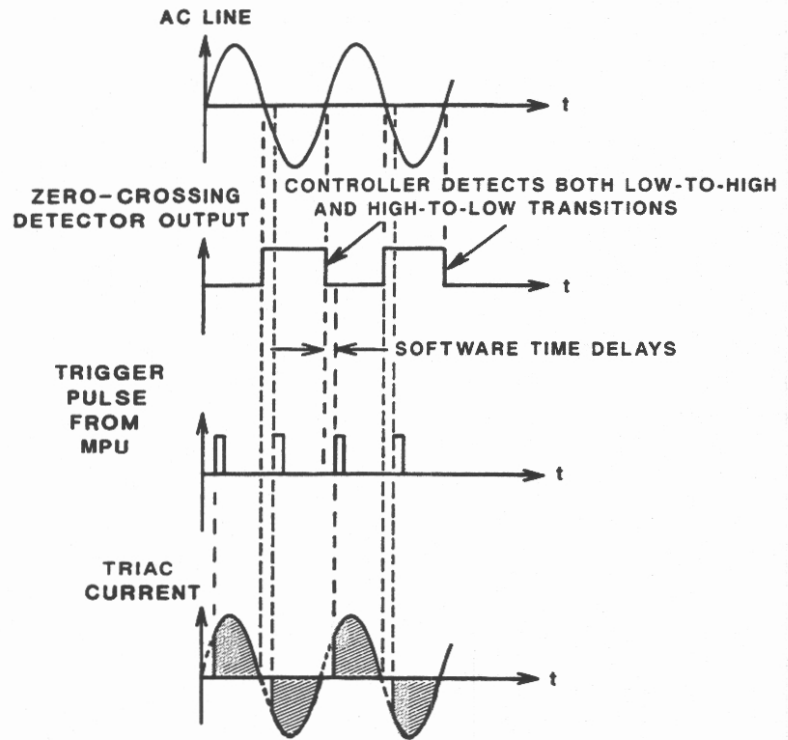


Figure 9-15

Timing diagrams for software control of an SCR.



**Figure 9-16**  
Timing diagrams for software control of a TRIAC.

The SCR or TRIAC triggering pulse can be generated via any output port line using the circuit shown in Figure 9-17. The pulse generated by the controller turns-on an optoisolator. An optoisolator is used since its input is TTL compatible. Even more important, the optoisolator provides isolation between the controller and line voltage circuits. This protects the controller circuits and reduces possible shock hazards, since the two circuit grounds are separate. When the optoisolator is turned on, its output phototransistor conducts and produces the required triggering pulse to the SCR or TRIAC. The optoisolator only needs to be turned-on for a short period of time. Recall that once an SCR or TRIAC is triggered, it will conduct throughout the half-cycle even after the triggering pulse is removed.

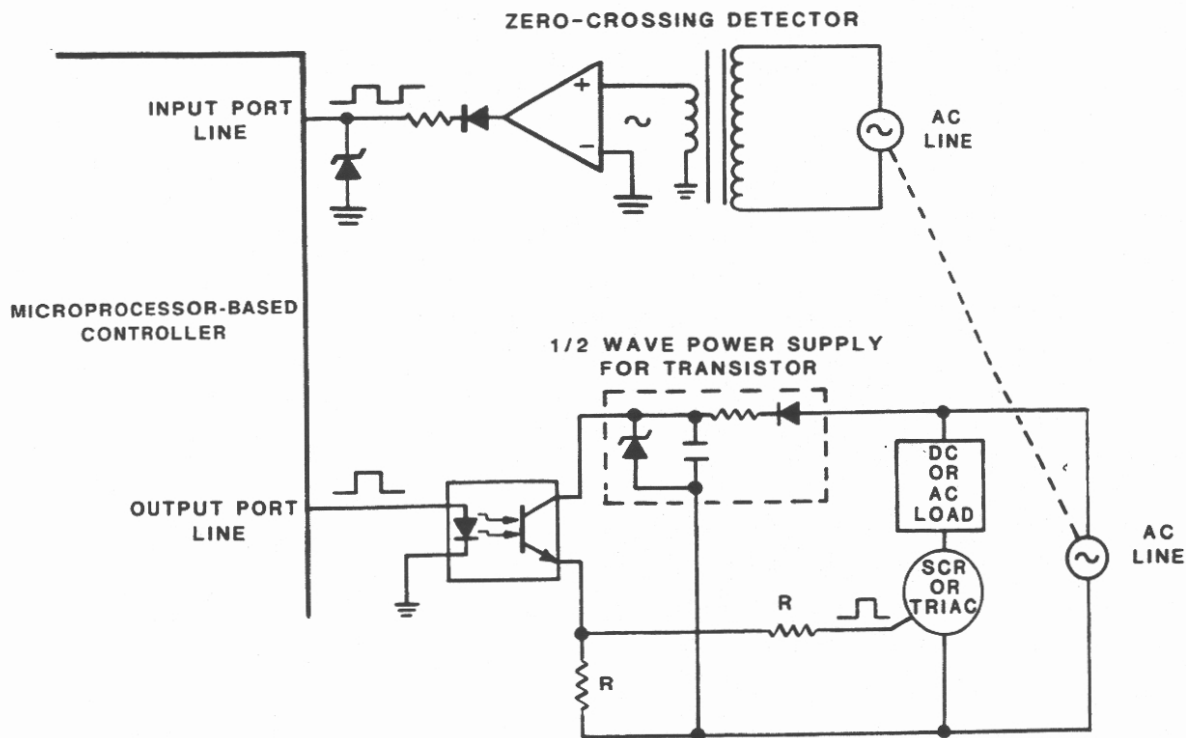


Figure 9-17

A computer-based SCR/TRIAC control circuit.

Notice that the components at the top of the SCR/TRIAC control circuit form a small power supply to bias the output transistor of the optoisolator from the AC line source. Of course, the microprocessor controller power supply could be used, but this would defeat the purpose of the optoisolator, since there would be no electrical isolation between the two circuits.

The circuit just described can be used to control high power DC or AC loads. An SCR must be used for DC loads and a TRIAC for AC loads. Possible loads include heating/cooling systems, lighting systems, DC motors, and AC motors. The heating/cooling level, brightness of a lamp, or speed of a motor is controlled by the amount of software time delay executed by the MPU. The obvious advantage of microprocessor-based control is that the MPU can sense temperature, light, or motor speed conditions and make a control decision. Once the control decision is made, the MPU simply executes the appropriate time delay to control the load.

## Relay and Solenoid Actuators

A relay is a device that makes or breaks an electrical current. As a result, a relay is often placed in series with a high power DC or AC load and used to open and close the electrical supply current to the load. The relay is required since the digital control circuit is not capable of directly handling the high current levels in the load circuit.

Relays can be subdivided into two general categories: **solid-state relays** and **electro-mechanical relays**. As its name implies, the solid-state relay is all electronic and has no moving parts. On the other hand, the electro-mechanical relay uses moving mechanical contacts.

## SOLID-STATE RELAYS

Several solid-state relays are pictured in Figure 9-18. Notice that each has two input terminals and two output terminals. The low power digital signal from a MPU control circuit can be applied directly to the input terminals as shown. An AC or DC load is then connected to the relay output terminal as shown. Notice that the relay is in series with the load and its electrical supply. Thus, the relay is used to open and close the supply current to the load. Put another way, the relay is actually turning the load on and off. The load is often a control device, such as a valve or motor, that provides direct control of an industrial process.

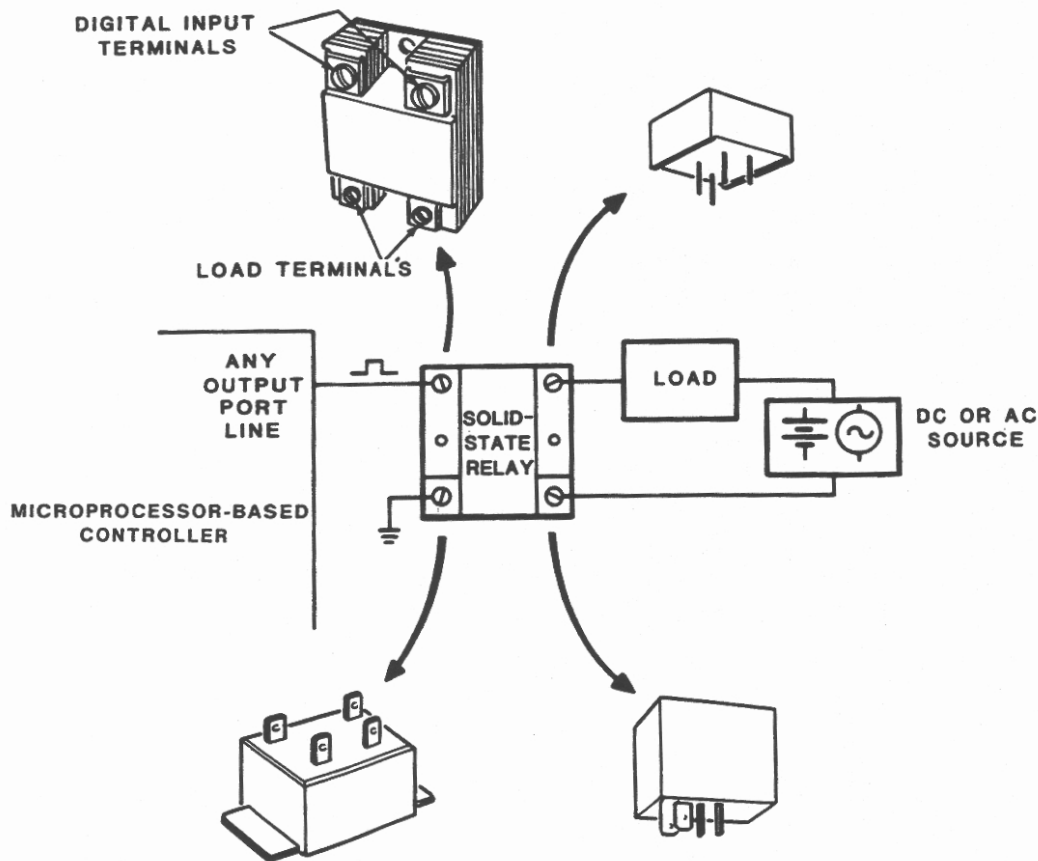


Figure 9-18  
Several typical solid-state relays.



Probably the most important feature of the solid-state relay is that its input is optically isolated from its output. Thus, like an optoisolator, the solid-state relay provides electrical isolation of several kilovolts between its input and output terminals. Solid-state relays are available to handle load currents up to 50 amperes. However, as you might guess, their cost is directly proportional to their current carrying capability. The output load voltage rating of a solid-state relay is usually specified as a range of voltages. For instance, a given AC relay might specify its output voltage rating as 24-140 volts (rms), while a second relay might specify a 48-280 volts (rms) load voltage rating. Clearly, the second relay could be used to control both a 120 volt and 240 volt AC load, while the first relay is limited to 120 volt loads. A typical load voltage rating for a DC relay is 3 to 50 volts.

When solid-state relays are used to control heavy AC loads, it is important that the relay not be energized during the peak of the AC cycle. If this happens, the instantaneous high current pulse might exceed the relay current rating and damage the relay. Special devices called zero-voltage switching solid-state relays are available for high AC current applications. These devices are used in the same way as standard solid-state relays. However, they include internal circuitry that delays the relay from turning-on until it detects the zero-crossing point of the AC voltage.

Another consideration for controlling high DC or AC current levels is heat sinking. The internal components of a solid-state relay dissipate a considerable amount of heat when conducting high current levels. If proper heat sinking is not provided, the solid-state relay will not conduct to its rated current level, and it could be damaged. Heat sinking usually involves mounting the relay on an aluminum heat sink block. Consult the manufacturer of the relay for the recommended heat sinking precautions.

Finally, if the solid-state relay is controlling a reactive load, such as a motor, the relay output should be protected with a **snubber circuit**. A typical RC snubber circuit is shown in Figure 9-19. This circuit dissipates any *back emf* voltage, or reverse voltage spike, generated by the reactive load when the relay is turned off. Without a snubber circuit, a back emf voltage spike could cause damage to the relay or, at the least, cause the relay to turn back on. Some solid-state relays have an internal snubber circuit. With these devices, no external components are required.

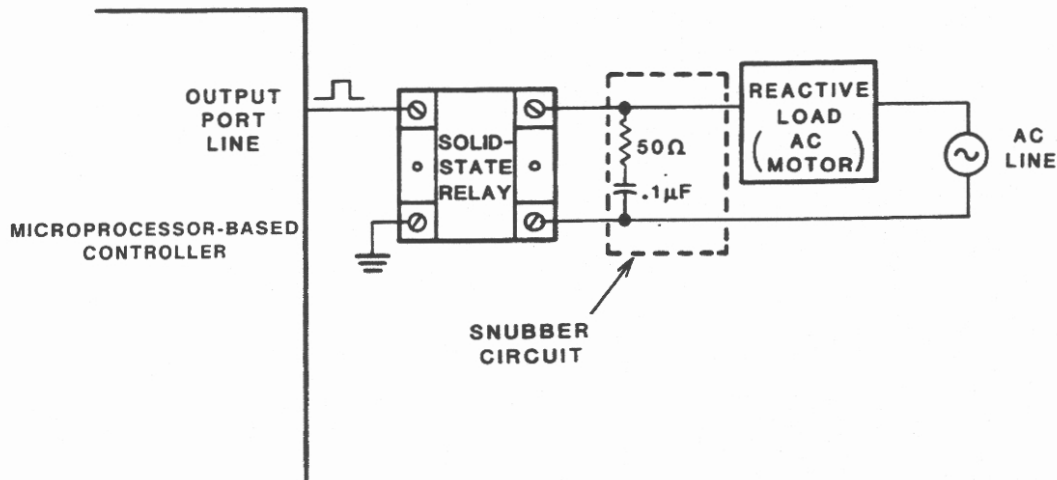


Figure 9-19

A snubber circuit should be used when controlling reactive loads, such as AC motors.

## ELECTROMECHANICAL RELAYS

Electromechanical relays are available in all shapes and sizes. Some typical relays are pictured in Figure 9-20. They range from small reed relays for low current applications to heavy duty power relays.

Small reed relays are usually sealed in a dual in-line package, or DIP, for mounting in standard IC sockets (Figure 9-20A). Reed relays are generally rated for less than one ampere of load current, with 0.1, 0.25, and 0.5 amps being typical current rating values. Load voltages up to 200 volts DC and 140 volts AC (rms) can be switched by many of these devices.

Input control voltage levels are usually 5, 12, and 24 volts. Consequently, a reed relay can be controlled using an open collector TTL output. Even with a 5 volt reed relay, an open collector device is required to provide sufficient current to energize the relay.

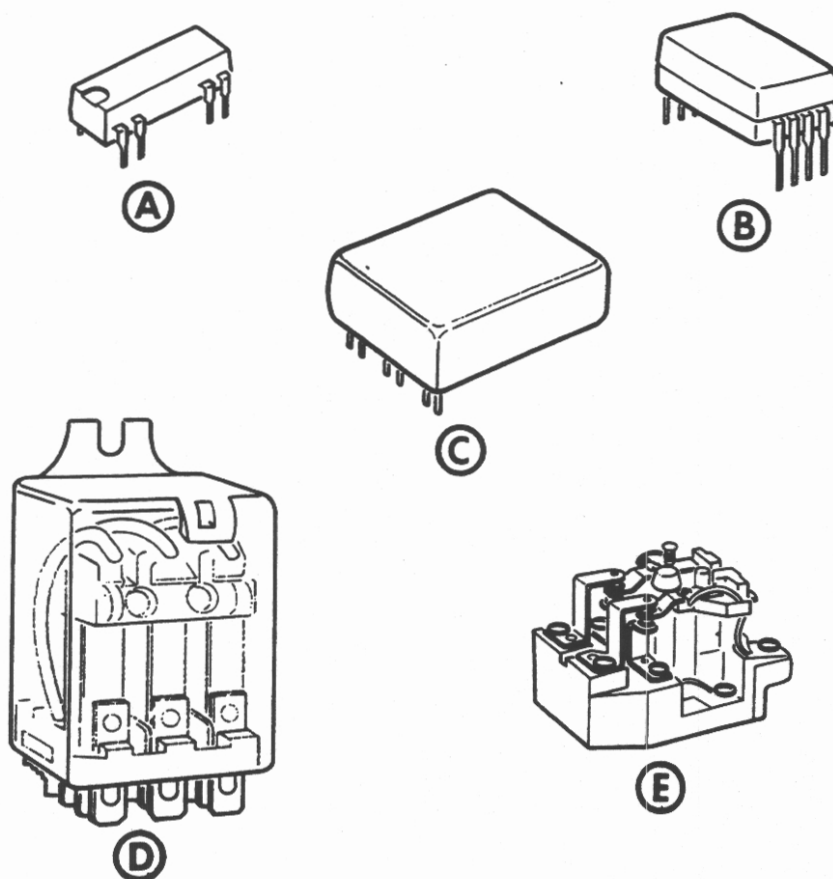


Figure 9-20  
Typical electromechanical relays.

The internal structure of a simple reed relay is shown in Figure 9-21. As you can see, the input side consists of a small coil. An optional internal diode in parallel with the coil will suppress any back emf generated when the coil is de-energized. Energizing the coil by application of the required DC input voltage level causes a small set of reed contacts to close (normally open) or to open (normally closed).

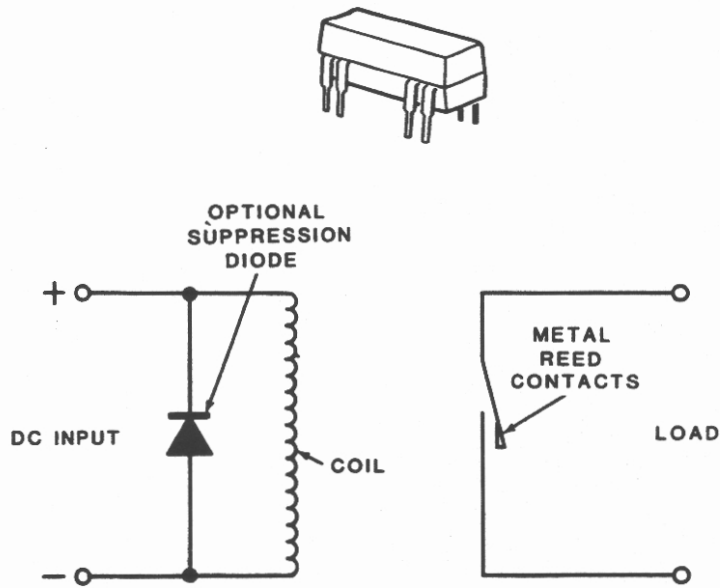


Figure 9-21  
Inside a reed relay.

Reed relays are available with several different contact configurations. A few of the more common configurations are illustrated by the contact diagrams in Figure 9-22. Notice that there is a standard code associated with each contact arrangement. For example, Form A is a normally open single-pole/single-throw, or SPST-NO, relay. On the other hand, a Form B relay is a normally closed single-pole/single-throw, or SPST-NC, relay. Other options provide for just about any possible contact arrangement.

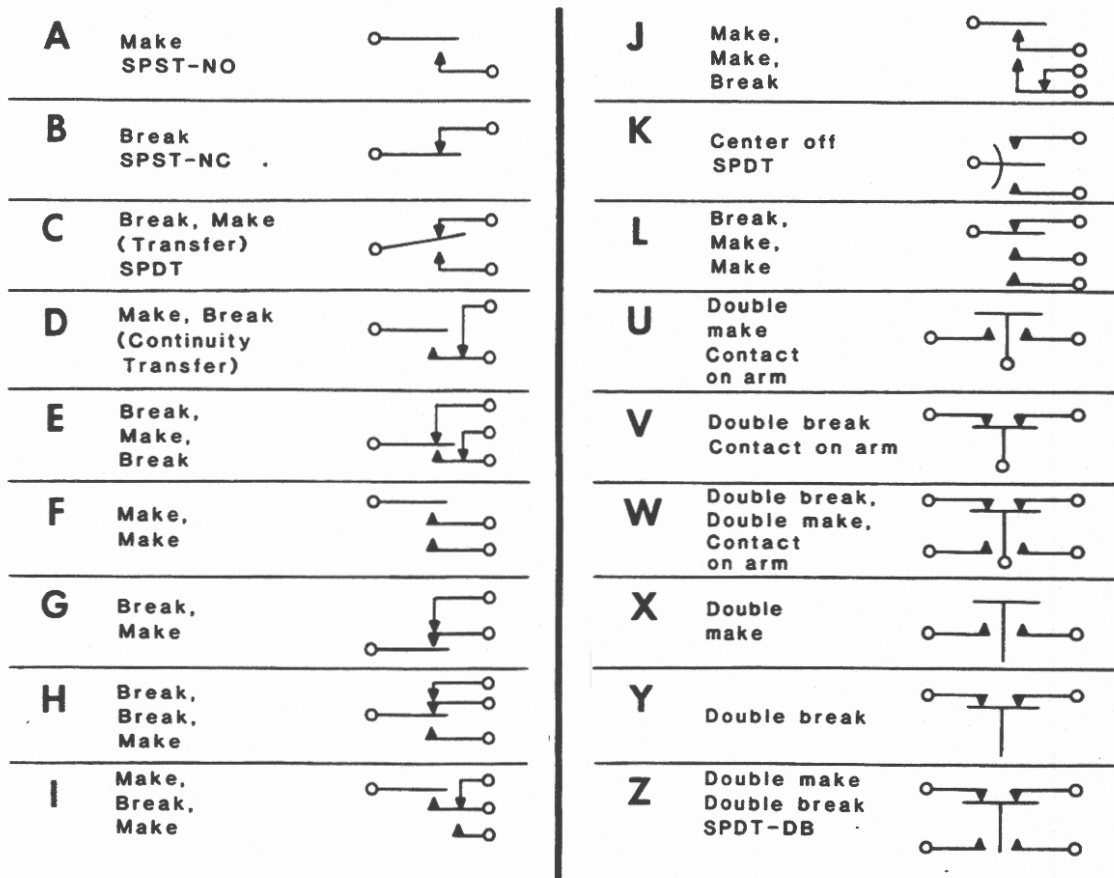


Figure 9-22

Reed relay contact configurations.

A high power relay must be used to control load currents above one ampere. High power electromechanical relays are available in a variety of power ratings and contact configurations to control both DC and AC current levels up to 50 amperes. Some power relays require DC excitation, while others require AC excitation. Since power relays cannot be controlled directly by digital signals, an intermediate solid-state relay must be used as shown in Figure 9-23. Notice that the solid-state relay is activated by the digital output. The intermediate solid-state relay then energizes the high power relay.

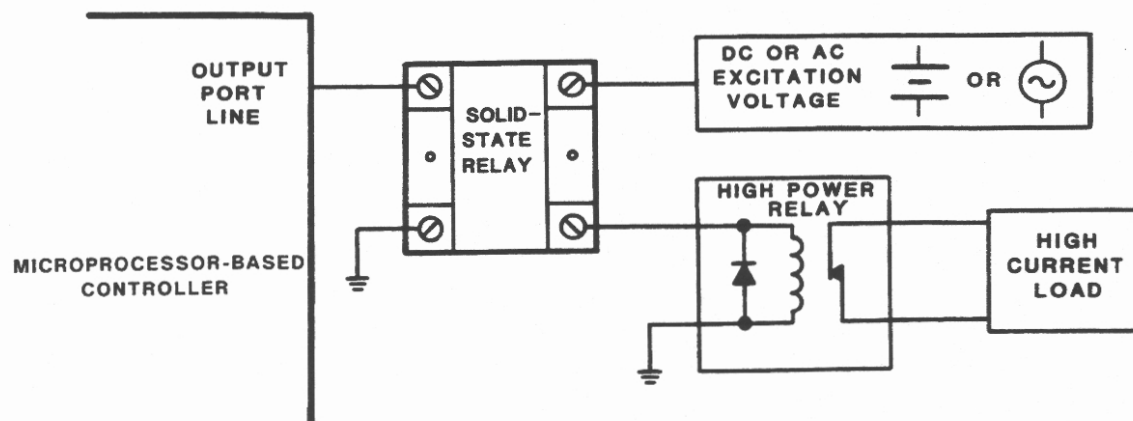


Figure 9-23

A solid-state relay is used to energize a high power relay under computer control.

## SOLENOIDS

A solenoid, pictured in Figure 9-24, is an electromechanical actuator that converts electrical energy into a linear mechanical motion. Solenoids are used in applications where a sudden force must be applied to an object. A solenoid consists of a plunger surrounded by a coil. The plunger is made of a ferrous material, such as hard steel, and may be free or spring-loaded. Application of an excitation current to the solenoid coil creates a magnetic field, which exerts a force on the plunger and causes it to move, or actuate.

There are both DC and AC solenoids. A DC solenoid requires a DC excitation current, while an AC solenoid requires an AC excitation current. Like heavy duty relays, solenoids must be energized using an intermediate solid-state relay for digital control. The digital circuit activates the intermediate solid-state relay which, in turn, passes the excitation current to the solenoid. This idea is shown in Figure 9-24.

Solenoids are rated for either temporary duty (intermittent) or continuous duty. Temporary duty solenoids can be energized for short periods of time. On the other hand, continuous duty solenoids can be energized for a longer period of time without damage.

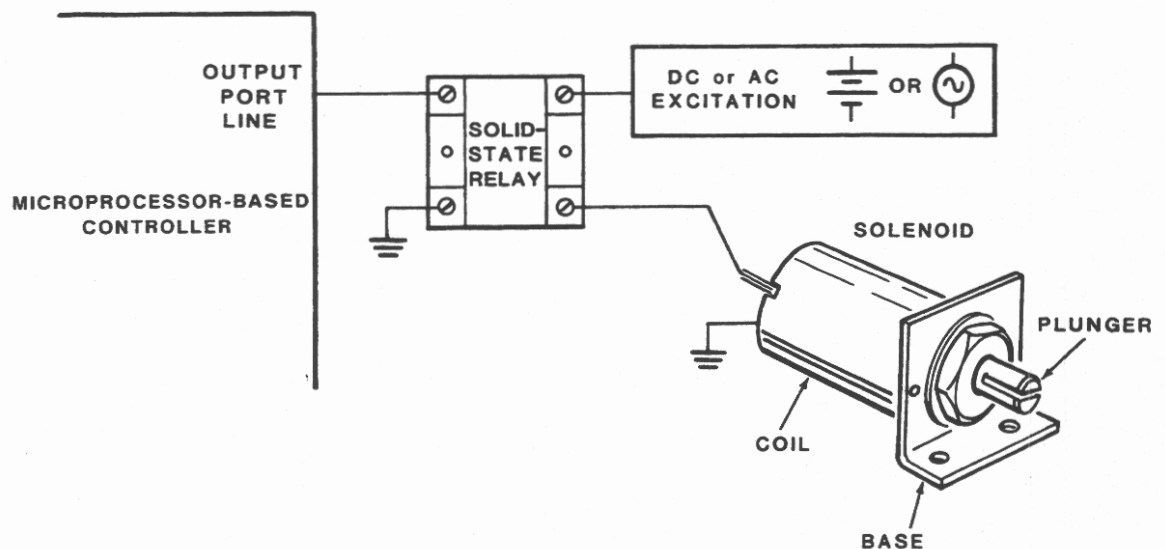


Figure 9-24

A solenoid and its associated computer-based control circuit.

## Self-Test Review

1. A small relay is rated at 25 V and requires 30 mA to be activated. Which of the open collector devices listed in Table 9-1 could be used to drive this relay?
2. A logic \_\_\_\_\_ must be applied to the base of an NPN control transistor for it to conduct.
3. Why are transistor arrays sometimes preferred over multiple discrete power transistors?
4. Power MOSFETs have the advantage of a \_\_\_\_\_ over bipolar power transistors.
5. An integrated device that includes both digital logic and driver transistors is the \_\_\_\_\_.
6. The two electrical components that make up an optocoupler are the \_\_\_\_\_ and \_\_\_\_\_.
7. State the most important feature of an optocoupler.
8. Two common thyristor control devices are:  
\_\_\_\_\_.  
\_\_\_\_\_.
9. True or False: TRIACs are generally used to control AC loads.
10. The point at which an SCR or TRIAC is triggered on the AC source waveform is called the \_\_\_\_\_.
11. What determines the firing angle of an SCR or TRIAC in a computer control circuit?
12. Name the two general categories of relays.  
\_\_\_\_\_.  
\_\_\_\_\_.



13. List three things that must be considered when controlling heavy/reactive AC loads with a solid-state relay.

\_\_\_\_\_.

\_\_\_\_\_.

\_\_\_\_\_.

14. A single-pole, double-throw (with center position off) switching application would require a type \_\_\_\_\_ reed relay.
15. An electromechanical device that converts electrical energy into linear motion is the \_\_\_\_\_.

## Answers

1. The 7406 hex inverter, since the relay ratings are within the maximum sink current and voltage ratings of the device.
2. 1.
3. Transistor arrays are sometimes preferred over multiple discrete power transistors because the transistors within the array have very close electrical and thermal properties. As a result, they have very similar operating characteristics.
4. Very high input impedance.
5. Peripheral Power Driver.
6. IRED.  
Phototransistor.
7. The most important feature of an optocoupler is the electrical isolation that it provides between its input and output.
8. Two common thyristor control devices are the SCR and TRIAC.
9. True: TRIACs are generally used to control AC loads.
10. Firing Angle.
11. The amount of software time delay between zero-crossing and gate triggering determines the firing angle of an SCR or TRIAC in a computer control circuit.
12. Solid State.  
Electromechanical.
13. When controlling AC loads with a solid-state relay, you must:
  - A. Make sure that the relay is not energized during the peak of the AC cycle. This requires a zero-voltage switching solid-state relay.
  - B. Make sure that you have provided heat sinking according to the manufacturer's recommendations.
  - C. Use a snubber circuit to dissipate any back emf generated by reactive loads.
14. K type. (See Figure 9-22)
15. Solenoid.

## **MPU CONTROL OF DC MOTORS**

In this section, you will learn how to control the speed and direction of DC motors. You should be cautioned however, that motor control is a science in itself. In fact, a complete course could be written solely on motor control and positioning. In this section, the intention is to touch on the basics of motor control. This material, coupled with the position sensing material presented in Unit 8, should give you the necessary tools required to design simple motor control circuits. At a minimum, the material presented here will provide you with a fundamental understanding of motor control principles and prepare you for further study in this field.

### **Controlling Permanent Magnet DC Motors**

Permanent magnet DC motors are easily controlled using one of two techniques: A D/A converter or pulse-width modulation. Let's take a closer look at each method, since they are both very common in the industry.

#### **USING A D/A CONVERTER FOR MOTOR CONTROL**

In Unit 6, you learned how the direction and speed of a DC motor is controlled with the analog output of a D/A converter, or DAC. The speed of rotation of the motor is controlled by the amount of control voltage applied to the motor. A bipolar DAC is used to allow variable speed control in both directions of rotation. One output polarity will cause the motor to rotate in a given direction. The opposite output polarity will reverse the direction of rotation. The speed of rotation is controlled, or adjusted, by the digital value written to the DAC via an output port of a microprocessor-based controller.

## USING PULSE-WIDTH MODULATION FOR MOTOR CONTROL

Look at the digital TTL pulses in Figure 9-25. As you know, the duty cycle of this signal is the ratio of the time the signal is high ( $T_H$ ) to the period for one cycle, or:

$$\text{Duty Cycle} = \frac{T_H}{\text{Period}} \times 100\% \\ (\text{in } \%)$$

From the above relationship, it is easy to see that the longer the signal is high, the larger the duty cycle. In addition, the average DC voltage produced by the signal increases as the high-time increases. Thus, the *average* DC voltage is directly proportional to the duty cycle. For example, if the duty cycle were 0%, the average DC voltage would be 0 V. At the other extreme, if the duty cycle were 100%, the average DC voltage would be 5 V. In fact, the average DC voltage produced by the TTL signal could be calculated for any duty cycle using the following equation:

$$V_{DC} = 5 \text{ V} \times \text{Duty Cycle}$$

When using this equation, the duty cycle is expressed as a decimal value, rather than a percent value.



$$\text{DUTY CYCLE} = \frac{T_H}{T} \times 100\% \\ V_{DC} = 5 \text{ V} \times \text{DUTY CYCLE}$$

Figure 9-25

Pulse-width modulation uses the principle that the average DC voltage produced is proportional to the duty cycle of a control signal.

**Example 9-1:**

A 1 kHz TTL signal is high for .2 ms of its period.

- A. What is the duty cycle of the signal?
- B. What is the average DC voltage produced by the signal?

**Solution:**

- A. The period of a 1 kHz signal is 1/1000 Hz, or .001 sec. This is equivalent to 1 ms. Thus, the duty cycle is:

$$\begin{aligned}\text{Duty Cycle} &= \frac{T_H}{\text{Period}} \times 100\% \\ &= \frac{.2 \text{ ms}}{1 \text{ ms}} \times 100\% \\ &= .2 \times 100\% \\ &= 20\%\end{aligned}$$

- B. Using this duty cycle and the above voltage equation, the average DC voltage is:

$$\begin{aligned}V_{DC} &= 5 \text{ V} \times \text{Duty Cycle} \\ &= 5 \text{ V} \times .2 \\ &= 1 \text{ V}\end{aligned}$$

Now, let's look at the theory behind **pulse-width modulation** or **PWM**. If you change the duty cycle of a digital signal, you can change the average DC voltage produced by the signal. Then if the signal is applied to a DC motor control circuit, the motor "sees" the average voltage level. Thus, the speed of the motor can be controlled by changing the duty cycle of the digital control signal. The beauty of doing this is that no D/A converter is required. The motor is controlled directly from the digital control signal.

A simple motor control circuit that uses this idea is shown in Figure 9-26. Here, two output port lines from a microprocessor-based controller are used to supply two digital control signals, A and B. Signal A is used to control the motor speed in one direction, while signal B is used to control the motor speed in the opposite direction.

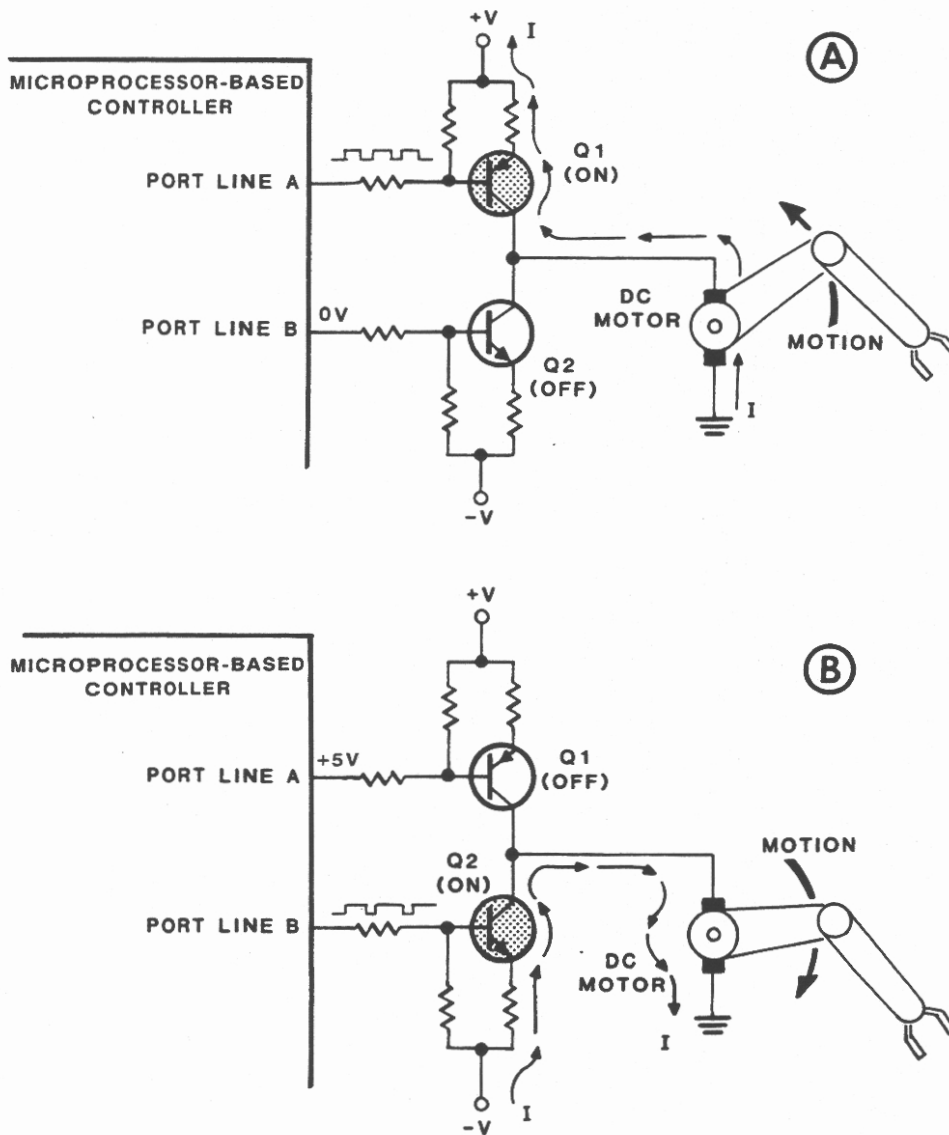


Figure 9-26

A PWM control circuit using bipolar control transistors.

Let's look at how this is done. Suppose that signal B is 0 V and signal A is the active control signal. This turns off Q2 (since Q2 is an NPN transistor) and turns on Q1, as illustrated in Figure 9-26A. Here, electron current flows from ground, up through the motor, and through Q1 to the +V supply. The amount of current flow depends on the amount of conduction through Q1, which is controlled by the duty cycle of control signal A. Consequently, the speed of the motor is directly proportional to the duty cycle of the control signal.

Next, suppose that signal A is +5 V and signal B is the active control signal. This turns off Q1 (since Q1 is a PNP transistor) and allows Q2 to conduct in proportion to the duty cycle of signal B as illustrated by Figure 9-26B. Now electron current flows from the negative supply (-V), up through transistor Q2, and down through the motor to ground as shown. This causes the motor to rotate in the opposite direction with a speed that is proportional to the duty cycle of control signal B. Of course, Q1 and Q2 are bipolar power transistors that are capable of conducting the current levels required by the motor.

Another PWM motor control circuit is shown in Figure 9-27. Notice that this circuit uses four power MOSFET control transistors that are arranged in an H-switch configuration. This configuration eliminates the need for a -V supply voltage.

The basic control idea is the same. Suppose that signal B is low, or 0 V, and signal A is the active control signal as shown in Figure 9-27A. In this case, Q2 and Q3 are turned-off, while Q1 and Q4 conduct in proportion to the duty cycle of control signal A. This creates an electron current flow from ground, up through Q4, up through the motor, and up through Q1 to the +V supply.

Next, suppose that control signal A is low and signal B is the active control signal. This turns-off Q1 and Q4, while allowing Q2 and Q3 to conduct in proportion to the duty cycle of control signal B as illustrated by Figure 9-27B. Now electron current will flow from ground, up through Q2, down through the motor, and up through Q3 to the +V supply as shown.

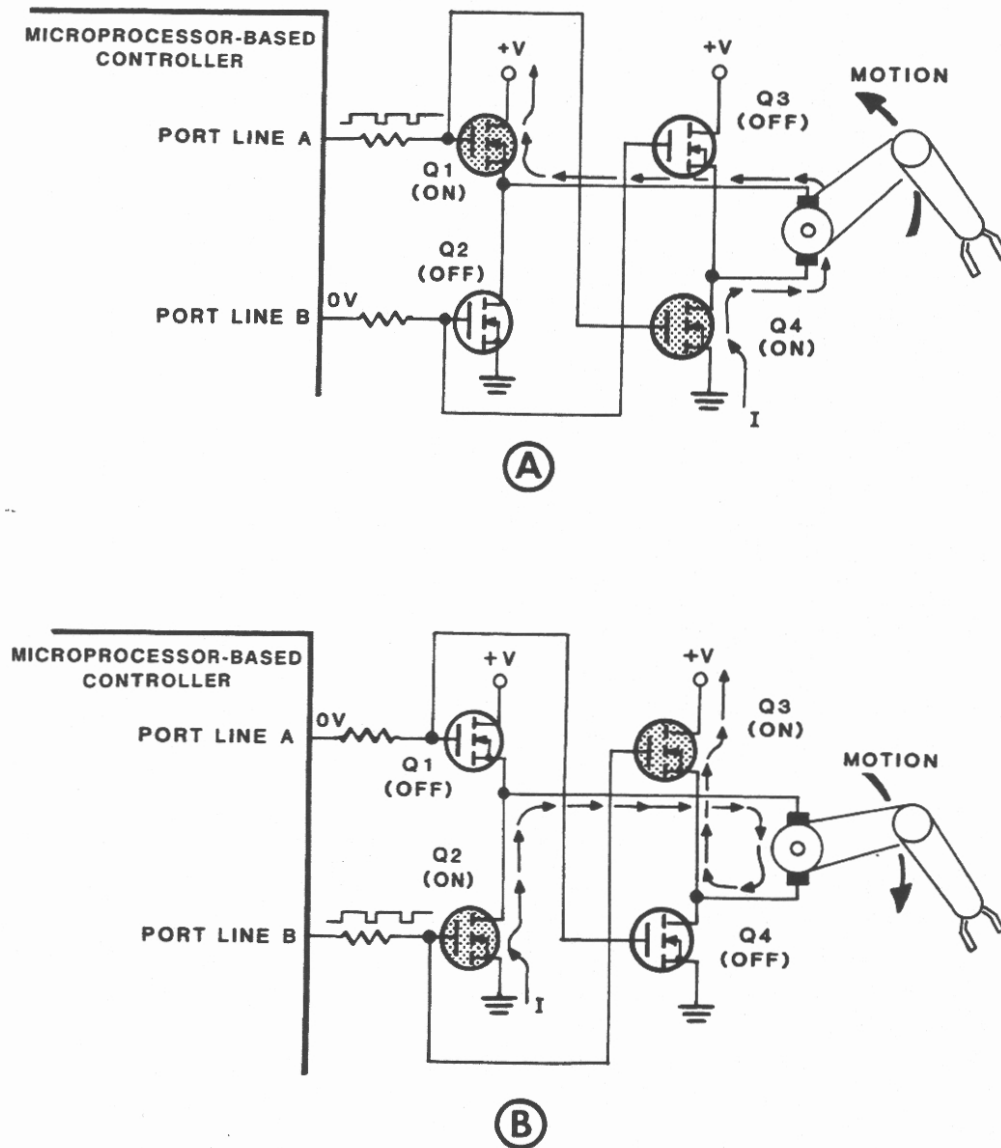


Figure 9-27

A PWM control circuit using power MOSFETs in an H-switch configuration.

The advantage of pulse-width modulation, or PWM, is the simplicity of the computer interface. The digital control signal does not have to be converted to analog for control purposes. Rather, the motor is controlled directly by a digital control signal via two or four power transistors. Of course, control software must be written to change the duty cycle of the control signal, as required, to change the motor speed and direction. We should mention that pulse-width modulation can also be adapted to controlling wound-field motors using these same control principles.



## CONTROLLING WOUND-FIELD DC MOTORS

As you are aware, you can control the speed of small permanent magnet type DC motors by simply varying the motor supply voltage. The resulting speed is directly proportional to the excitation voltage level. However, if larger torques are required, you must use a wound-field DC motor. Speed control of wound-field DC motors is different from that of permanent magnet type DC motors. The most common method used to control the speed of a wound-field DC motor is the shunt-field control method. Obviously, this method can only be used with shunt or compound motors.

Shunt-field speed control is obtained by reducing the shunt-field current of a shunt or compound motor. Reducing the shunt-field current reduces the magnetic flux of the field circuit, which causes the motor speed to increase. The speed increases since a counter emf generated in the armature circuit decreases. The *counter emf* is produced by generator action between the armature coils and field flux. By Lenz's law, the counter emf produces a current which opposes the armature current. Consequently, as the counter emf is reduced, the net armature current increases, thereby causing the motor speed to increase. Conversely, increasing the field current increases the counter emf which, in turn, causes the motor speed to decrease. The field current is made to vary by inserting a variable resistance in the field circuit.

Shunt-field speed control provides a relatively smooth and efficient technique of varying the speed of a shunt or compound motor over its operating range. However, shunt-field control is only suitable for obtaining speeds greater than the base, or rated operating speed. You can obtain a momentary speed reduction below base speed by over-exciting the field, but prolonged over-excitation will overheat the motor. In addition, magnetic saturation of the field only permits a slight reduction in speed for large increases in field excitation. Consequently, the maximum speed range using shunt-field control is usually 4:1. The curve in Figure 9-28 shows the relationship between speed and field current for a typical shunt or compound motor.

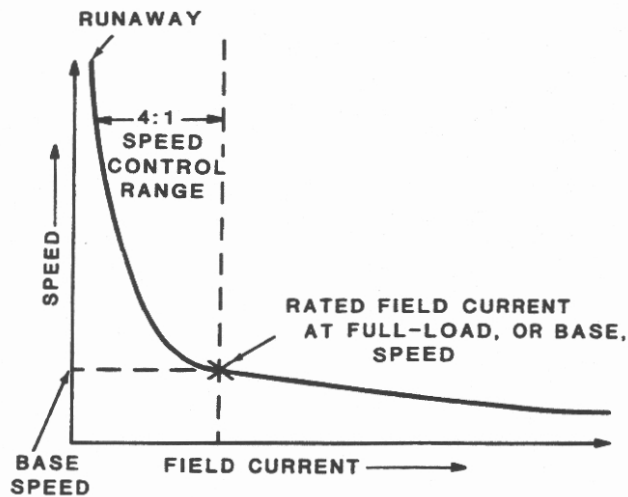


Figure 9-28

A variable resistance can provide a 4:1 speed control range by changing the field current in a shunt motor. Notice that speed increases as field current decreases.

The voltage supplied to the armature circuit can be varied to change the speed of a DC motor. This is called **armature-voltage control** and is used to vary the motor speed downward from its base speed. Variation of the armature voltage will typically provide a 10:1 speed range adjustment below the base, or rated, speed of the motor. Many times, shunt-field control is used to provide speed control above the base speed of the motor and armature-voltage control is employed to provide control below the base speed of the motor.

One major problem encountered with controlling motor speed is heat dissipation. Most DC motors are cooled by their own self-ventilating action. As motor speed is lowered, the ventilation capability is also reduced. At a given low speed, increased loads cause higher armature current levels, which generate heat that must be effectively dissipated to prevent motor damage.

Unlike permanent magnet DC motors, wound-field DC motors do not lend themselves as easily to microprocessor-based control. From the above discussion, you are aware that variable speed control of a wound-field motor requires that the current in field, or voltage applied to the armature circuit, be controlled. One method is to use an SCR to control the average power supplied to the motor as described earlier in this Unit. Another way that microprocessor-based control can be accomplished is by using a solid-state voltage controlled variable resistance device like a power FET or MOSFET device as shown in Figure 9-29. Here, the output from a computer controlled DAC circuit is applied to the gate of a power MOSFET device. The power MOSFET is connected into the field circuit of a shunt-wound motor. As the voltage output of the DAC is varied by the controller, the MOSFET will conduct more or less current, thereby controlling the speed of the motor. Power MOSFETs are available to handle current levels up to 15 amperes, with breakdown voltages exceeding 200 volts. Of course, pulse-width modulation could also be used to eliminate the need for a DAC.

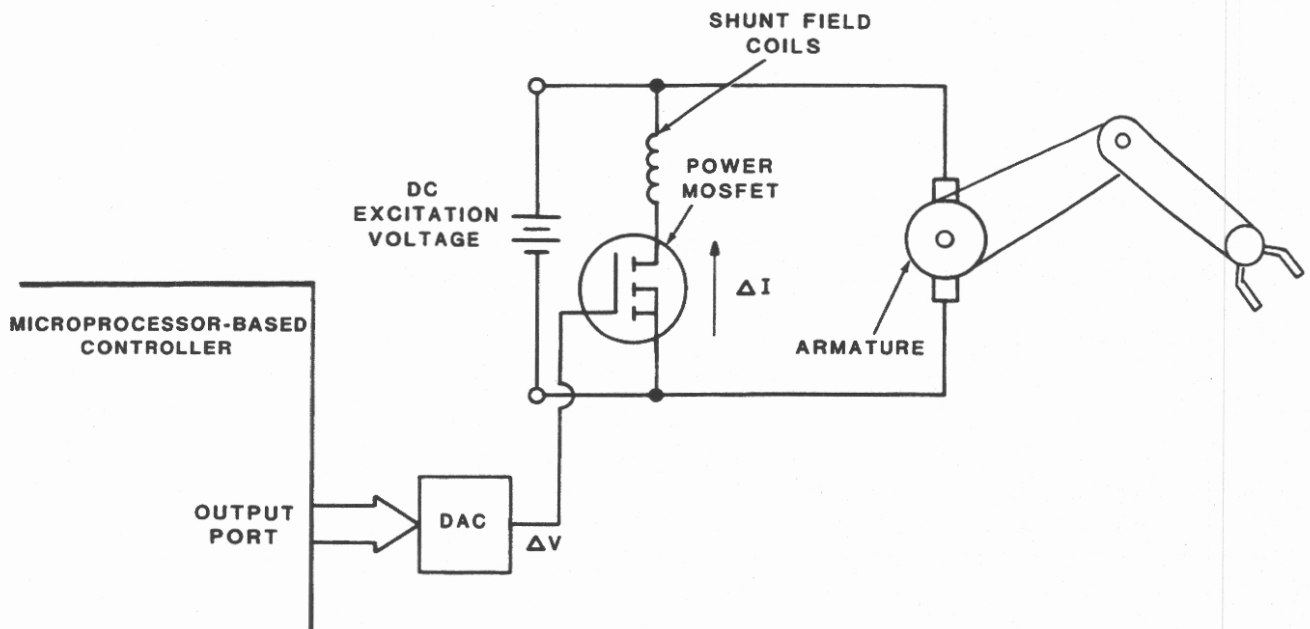


Figure 9-29

Using a power MOSFET to control a shunt-wound DC motor.

## Stepper Motors

Stepper motors are becoming increasingly popular in industrial applications, especially robotics, where computer control is the norm. They offer significant advantages over other types of DC motors when employing computer-based control.

A stepper motor is a motor that can rotate in either direction, start or stop at various mechanical positions, and move its rotor in precise angular increments for each input excitation change or step. The precise angular movement is repeated for each input step command, which results in the motor's ability to accurately position its rotor in a known repeatable direction.

Rotor position, speed, direction, and distance of angular travel are easy to control in a stepper motor. Since each input step moves the rotor to a known position, the only rotor error is the single step accuracy of the motor, regardless of distance of travel. This accuracy is typically 5% of one step, but can be less than 1% in some stepper motors. The number of steps required to complete one revolution of the rotor varies, depending on the specific motor and its intended application.

Stepper motors are typically available in the steps-per-revolution sizes shown in Figure 9-30. Each stepper motor is built for a specific step angle. However, they may be operated at one-half the given step angle, but at reduced torque.

| STEPS-PER-REVOLUTION | INCREMENTAL<br>ROTOR ANGLE<br>PER STEP |      |
|----------------------|----------------------------------------|------|
|                      | 240                                    | 1.5° |
|                      | 180                                    | 2.0° |
|                      | 144                                    | 2.5° |
|                      | 72                                     | 5.0° |
|                      | 48                                     | 7.5° |
|                      | 24                                     | 15°  |
|                      | 12                                     | 30°  |

Figure 9-30  
Typical stepper motor sizes.

## BIPOLAR PERMANENT MAGNET STEPPER

The bipolar permanent magnet stepper contains a permanent magnet rotor whose operation is a result of magnetic characteristics: like poles repel and unlike poles attract. The permanent magnet rotor, shown in Figure 9-31A, is an axially-oriented permanent magnet with a gear-like hub on each end. The north pole has teeth that are  $180^\circ$  out of phase from the south pole. The stator poles, shown in Figure 9-31B, also have teeth; but in this case, the magnetic poles are generated by the stator windings.

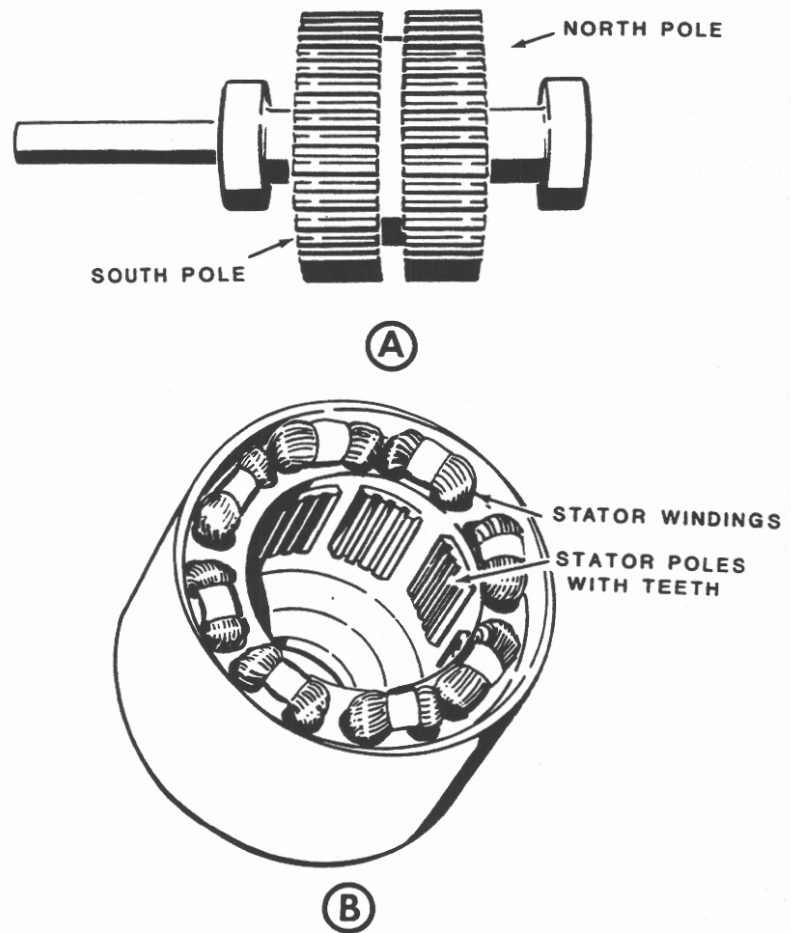


Figure 9-31

Construction of a permanent magnet stepper motor.

*The main idea behind its operation is that the number of teeth on the rotor is different from that of the stator. This is necessary so that the teeth of the rotor will never be lined up exactly with the teeth of the stator. It is this characteristic that actually determines the predictable movement of the rotor, since there is a*

magnetic attraction between the closest rotor and stator tooth. Moreover, the number of teeth on the rotor and in the stator will determine the amount of angular movement of the rotor each time the motor is stepped. The greater the number of teeth, the smaller the step angle. In addition, the magnetic relationship between the permanent magnet rotor and the stator produces some low-torque holding power, even when not energized. This characteristic is referred to as **residual torque**, and is a result of residual magnetism within the motor.

For ease of explanation, the permanent magnet stepper shown in Figure 9-32 has four stator windings and poles, labeled A, B, C, and D. The axially-oriented rotor consists of six permanent magnetic north and south poles, labeled N1, N2, N3, and S1, S2, S3 respectively.

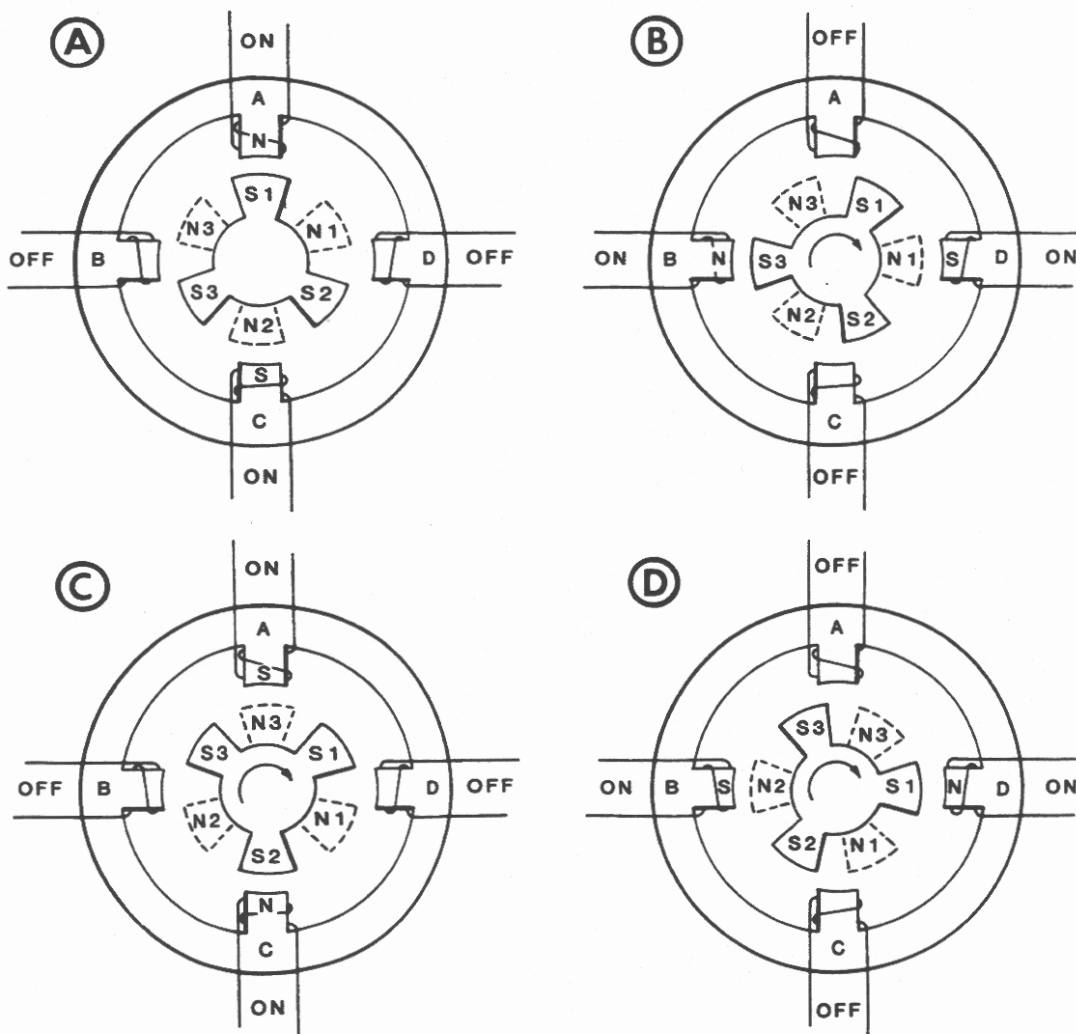


Figure 9-32

Operation of a bipolar permanent magnet stepper, using 4-phase excitation.

The excitation sequence (in this example, 4-phase excitation) of the windings is fairly simple. Consider the poles of the rotor, as shown in Figure 9-32A. If stator pole A is energized as a north pole and stator pole C is energized as a south pole (with stator poles B and D not energized), rotor pole S1 will align itself with stator pole A, and rotor pole N2 will align itself with stator pole C. If stator pole B is then energized as a north pole and stator pole D energized as a south pole (with stator poles A and C not energized), the rotor will rotate clockwise with rotor poles S3 and N1 aligning themselves to stator poles B and D respectively, as shown in Figure 9-32B.

To further step the motor, the current is reversed from its original direction in stator poles A and C, now making pole A a south pole and pole C a north pole. Since power has been removed from stator poles B and D, rotor poles S2 and N3 will align themselves to stator poles A and C, as shown in Figure 9-32C. To continue clockwise rotation, power is applied to stator poles B and D with the opposite polarity as before, resulting in rotor action as shown in Figure 9-32D. The next stepping action would be to apply power to stator poles A and C as in the original position (Figure 9-32A). Continuing this stepping action makes the rotor turn clockwise. For counterclockwise rotation, power would be applied to the stator windings in reverse order. That's all there is to it!

## BIFILAR, OR UNIPOLAR, MOTOR

A bifilar stepper motor, commonly referred to as a unipolar stepping motor, is simply a variation of the bipolar permanent magnet stepper motor just discussed. However, in the bifilar stepper, each stator winding has been center-tapped, which in reality places two stator windings on each stator pole. Since the stator windings are center-tapped, current can be sent through alternate halves of each winding to obtain alternate magnetic polarities.

The operation of a basic bifilar stepper motor is shown in Figure 9-33. For ease of understanding, the two phases are shown as four stator poles, and the rotor is shown with only a north and south pole. Actually, the bifilar stepper would have many more stator and rotor poles than this figure shows. Refer to step 1 of the sequencing chart in the figure. If stator pole windings A1 and B1 become north stator poles when energized, the rotor would assume the position shown in Figure 9-33A. In step 2 on the sequencing chart, stator windings A1 and B2 are on and stator winding B1 is off; this moves the rotor 90° clockwise to the position shown in Figure 9-33B. To position the rotor another 90° clockwise, as shown in Figure 9-33C, stator winding A1 is de-energized, stator winding A2 is energized, and stator winding B2 remains energized. This action corresponds to step 3 in the sequencing chart. For an additional 90° of rotor rotation, refer to the sequencing chart and observe Figure 9-33D. To complete one revolution of the rotor, the stator windings would be energized in the same manner as they were for step 1.

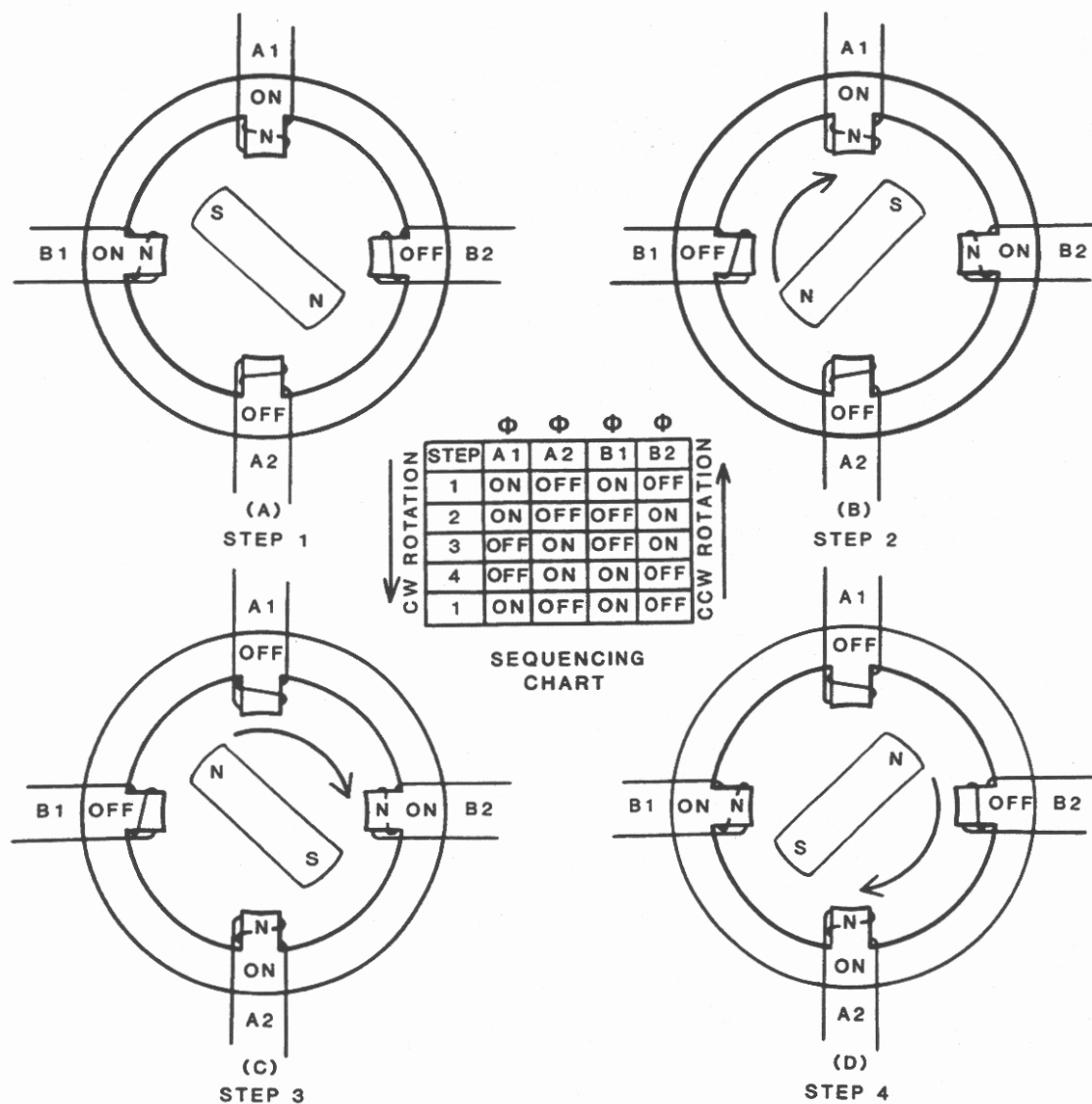


Figure 9-33

Operation of a 2-phase unipolar (bifilar) stepper motor.



For a bifilar motor to have the same number of turns per stator winding as a bipolar motor, the wire diameter must be decreased. Therefore, the stator resistance is increased. As a result, a bifilar (unipolar) stepper has about 30% less torque than a bipolar stepper at low step rates. In addition, the higher resistance stator windings of the bifilar produce a smaller inductive time constant in the stator field. Recall that the time constant of an inductive circuit is  $L/R$ , where  $L$  is the circuit inductance in henrys and  $R$  is the circuit resistance in ohms. As a result of this smaller time constant, the bifilar stepper motor has excellent high speed performance as compared to the bipolar motor. The relative speed/torque performance of a bifilar versus a bipolar permanent magnet stepper is illustrated by the graph in Figure 9-34.

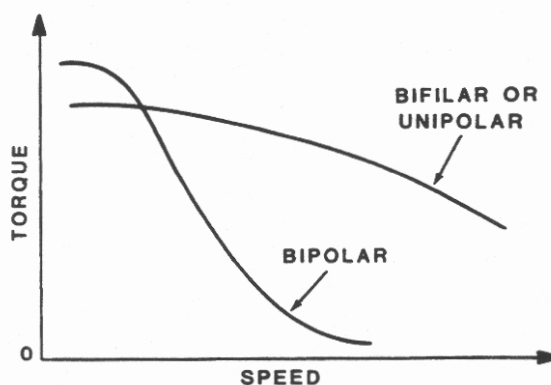


Figure 9-34

Speed/Torque performance for a bipolar versus a unipolar stepper motor.

## CONTROLLING STEPPER MOTORS

The performance of a given stepper motor in any particular application in terms of torque, speed, acceleration, step response time, and maximum stepping rates is as much a function of the control circuitry as the motor itself. The type of stepper motor being controlled (bipolar or unipolar), will determine what type of control circuit is used. This discussion will refer to the type of control as either bipolar control or unipolar control. As the name implies, the bipolar control circuit is used to control a bipolar stepper; while the unipolar control circuit is used to control a unipolar, or bifilar, stepping motor.

## Bipolar Control

The bipolar control circuit shown in Figure 9-35 is being used to drive a 2-phase bipolar permanent magnet stepping motor. As you recall, in a bipolar stepping motor each motor phase consists of only one stator winding per pole. Therefore, both ends of the winding must be alternately connected to the voltage source to produce the correct magnetic stator field. Hence, four transistors are required per phase, or a total of eight transistors for a 2-phase motor.

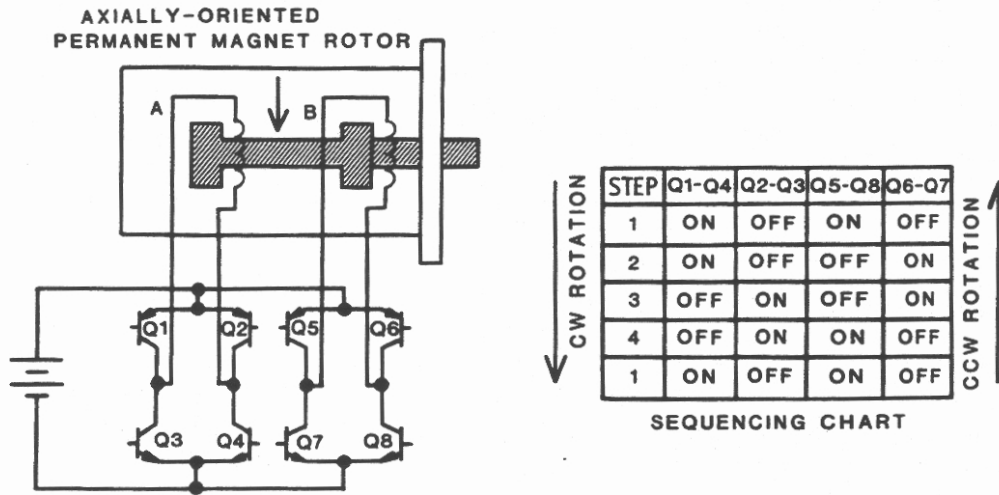


Figure 9-35  
Bipolar stepper motor control.

Referring to Figure 9-35, if transistor switches Q1, Q4, Q5, and Q8 were turned-on, electron current flow would be from the negative side of the battery through transistor Q4, up through winding A (creating a north pole), and through transistor Q1 to the positive side of the battery. In addition, current would also flow from the negative side of the battery through transistor Q8, up through winding B (creating a north pole), and through transistor Q5 to the positive side of the battery. This would cause the axially-oriented rotor to rotate clockwise because of the action of the magnetic fields. This corresponds to step 1 in the sequencing chart.

In step 2, stator winding A remains a north pole (transistor switches Q1 and Q4 are still on); but now, transistor switches Q5 and Q8 have been turned off and transistor switches Q6 and Q7 have been turned on. This switching action causes current to flow from the negative side of the battery through transistor Q7, down through winding B (now creating a south pole in winding B), and through transistor switch Q6 to the positive side of the battery. This causes a magnetic field that will step the rotor clockwise once again.

To obtain an additional clockwise step, transistors Q6 and Q7 remain on (maintaining a south pole in winding B), transistors Q1 and Q4 are now turned off, and transistors Q2 and Q3 are turned on. This causes current to flow from the negative side of the battery through transistor Q3, down through winding A (now creating a south pole in winding A), and through transistor Q2 to the positive side of the battery. This corresponds to step 3 on the sequencing chart.

To complete the stepping action, the conditions of step 4 on the sequencing chart must be met. In this case, transistors Q2 and Q3 remain on (maintaining a south pole through winding A), transistors Q6 and Q7 are turned off, and transistors Q5 and Q8 are turned on. Current now flows from the negative side of the battery through transistor Q8, up through winding B (again creating a north pole), and through transistor Q5 to the positive side of the battery. To continue the clockwise rotation, the stepping must continue in the sequence 1, 2, 3, 4, and so on.

## Unipolar Control

The unipolar control circuit shown in Figure 9-36 is used to drive a 2-phase bifilar (or unipolar) permanent magnet stepping motor. Keep in mind that, in a unipolar stepper, each phase consists of a center-tapped winding. The center-tap effectively divides each phase into two separate windings.

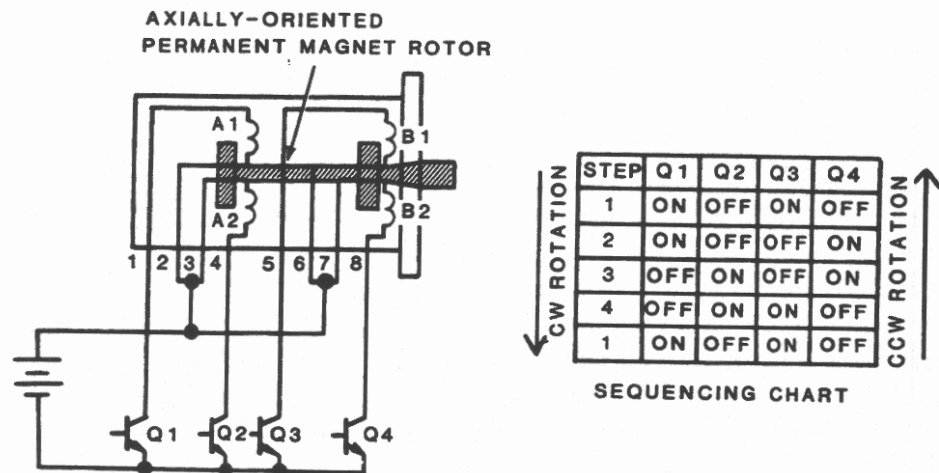


Figure 9-36

Bifilar, or unipolar, stepper motor control.

Let's look at how this works. The two halves of each stator coil in Figure 9-36 have been labeled A1, A2, and B1, B2, respectively. Assuming that the conditions of step 1 on the sequencing chart have been met, both transistor switches Q1 and Q3 are turned on. This allows electron current to flow from the negative side of the battery, through switch Q1 into terminal 1 of the motor, down through stator winding A1 (creating a south pole in stator A), and out of terminal 2 of the motor to the positive side of the battery. At the same time, electron current flows from the negative side of the battery, through switch Q3 into terminal 5 of the motor, down through stator winding B1 (creating a south pole in stator B), and out of terminal 6 of the motor to the positive side of the battery. This action creates a magnetic field that rotates the rotor clockwise one step.

Observing step 2 on the sequence chart, you see that switch Q1 is still on (creating a south pole through stator pole A), switch Q3 is off, and switch Q4 is now on. Now, current is allowed to flow up through stator winding B2; thus creating a north pole at stator pole B. Due to the movement of the magnetic field, the rotor moves another step clockwise. By studying the sequencing chart and the schematic diagram, it becomes readily apparent that steps 3 and 4 will rotate the rotor further clockwise.

The use of a bifilar stepper and unipolar control allows the drive circuit to be simplified. Not only are half as many power transistors required (4 versus 8), but timing is not as critical.

In either the bipolar or unipolar mode of control, you can make the rotor turn counterclockwise by simply reversing the stepping procedure shown in the sequencing chart. In other words, stepping sequence 1, 2, 3, 4, 1, etc. turns the rotor clockwise; while stepping sequence 1, 4, 3, 2, 1, etc. would turn it counterclockwise.

To improve the high speed performance of the unipolar control circuit, you can add series resistances as shown in Figure 9-37. The series resistance shortens the  $L/R$  time constant to allow a faster current, or magnetic field, build-up in the stator coils. As a result, the motor can be operated at higher speeds with increased torque. This arrangement is called a **series resistance limiting drive**. The price you must pay for better performance with this circuit is an increase in the supply voltage, since the series resistance drops the voltage supplied to the motor. As a rule of thumb, make the supply voltage 4 to 6 times the rated voltage of the motor. Then, pick a resistor value that will drop this supply level to the required motor voltage.

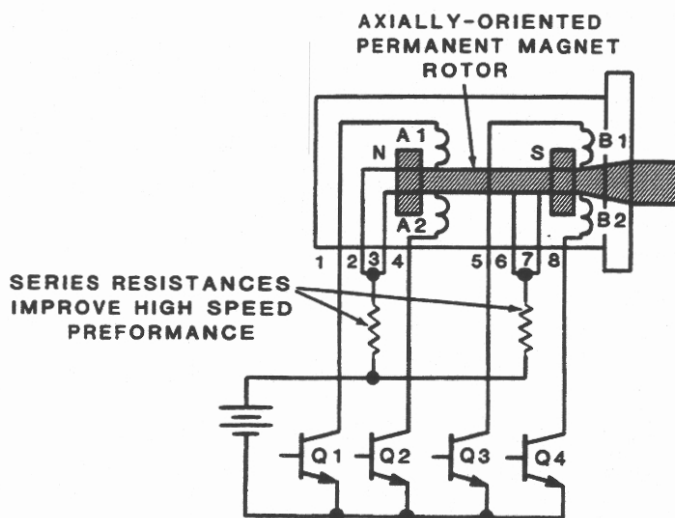


Figure 9-37

Adding series resistances improves the high speed performance of a unipolar stepper motor.

The control transistors must be low, medium, or high power transistors, depending on the size of the stepper motor. Power bipolar as well as power MOSFETs are common. However, there are three transistor characteristics you must consider when you are selecting a transistor for the stepper motor drive: the transistor current rating, reverse voltage rating, and switching time.

The transistor conducts its peak current levels when the motor coils are being energized and de-energized. The peak current levels are often twice the average DC current supplied to the windings. For this reason, you should pick a transistor that has a rating of at least 1 1/2 times the average current level supplied by the control circuit. Use Ohm's law to calculate the average current level by dividing the supply voltage value by the coil resistance plus any resistance in series with the coil. For example, suppose a bifilar stepper motor specifies a coil winding resistance of  $22\Omega$  and a rated operation voltage of 5 V. Then, without any added series resistance, the control circuit transistors must have a current rating of at least

$$1\frac{1}{2} \times \left( \frac{5\text{ V}}{22} \right), \text{ or } 340\text{ mA.}$$

The breakdown voltage rating of the control transistors should be five to ten times the rated motor voltage due to auto-transformer action within the motor windings. When a motor winding is de-energized, the collapsing magnetic field around the winding induces a transient back emf that can be many times the original supply voltage level. A control transistor must be capable of withstanding this reverse voltage spike while it is in its turned-off state. The transistor breakdown voltage rating is usually specified as  $BV_{CEO}$ .

In addition to picking a transistor with a high  $BV_{CEO}$ , a diode suppression circuit is often required. Three different diode suppression circuits are illustrated in Figure 9-38. In each illustration only one motor coil and its associated control transistor are shown. The diode suppression circuit must be repeated for each control transistor in the circuit. In any event, a single diode is placed across the coil windings in Figure 9-38A. When the coil is energized, the diode is reversed-biased and does not conduct. When the coil de-energizes, the back emf created by the collapsing magnetic field forward-biases the diode and allows the current flowing in the winding to dissipate within the diode/coil loop and not be seen by the transistor. The maximum reverse voltage that the transistor will see is the supply voltage plus the forward voltage drop of the diode (.3 V for germanium diode or .7 V for a silicon diode).

In Figure 9-38B, a resistor is added in series with the suppression diode. This shortens the  $L/R$  time constant and allows the reverse current in the winding to dissipate more rapidly when the coil is de-energized. The advantage is increased high speed performance. However, the transistor sees a larger reverse voltage. The reverse voltage spike now becomes the supply voltage plus the forward diode voltage, plus any voltage dropped across the resistor.

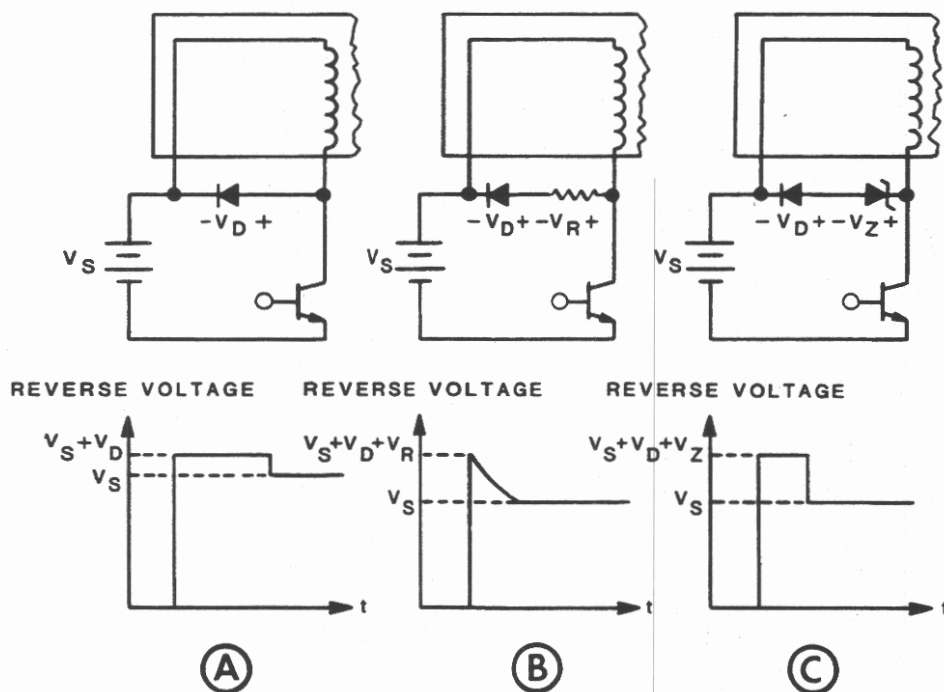


Figure 9-38

- A. Simple diode suppression.
- B. Diode resistor suppression.
- C. Zener diode suppression.

In Figure 9-38C, a zener diode is added in series with the suppression diode. This is a compromise between the single diode and diode/resistor arrangement shown in Figure 9-38A and B. The maximum reverse voltage that the transistor sees is now the sum of the supply voltage, plus the forward diode voltage, plus the rated zener voltage. The reverse voltage produced by this circuit is less than the reverse voltage produced by the diode/resistor arrangement in Figure 9-38B. In addition, the zener diode circuit allows the reverse current in the winding to dissipate more rapidly than the single diode circuit shown in Figure 9-38A; hence a compromise.

Finally, the on/off switching time of the transistor must be at least 2 microseconds to avoid wasting power and to keep the heat sinking to a minimum. Longer on/off switching times create more power dissipation in the form of heat within the transistor. A fast switching transistor will dissipate less heat and, therefore, require less heat sinking.

## The MPU Interface

Interfacing stepping motors to a microprocessor-based control system is relatively simple. The lines of an output port are used to drive the control transistors as illustrated in Figure 9-39. The diodes shown in Figure 9-39 are used to protect the port from the high motor voltages in case a control transistor should short-out. The resistors are used to limit the current through the control transistors.

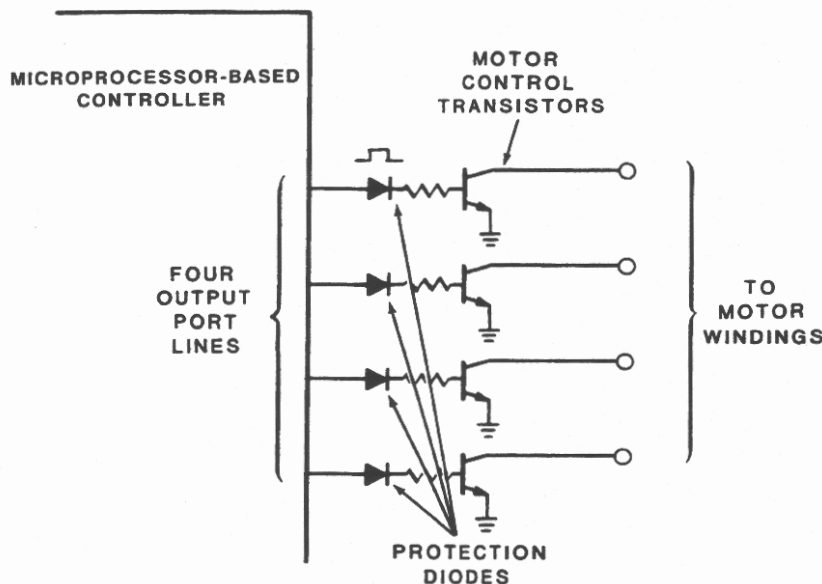


Figure 9-39

An output port can be used to drive the control transistors directly.



Excellent isolation is provided by the optoisolator circuit shown in Figure 9-40. Here, the optoisolators provide isolation of several kilovolts between the motor and MPU circuits. In addition, possible lethal situations are avoided due to the ground isolation provided by the optoisolator circuit.

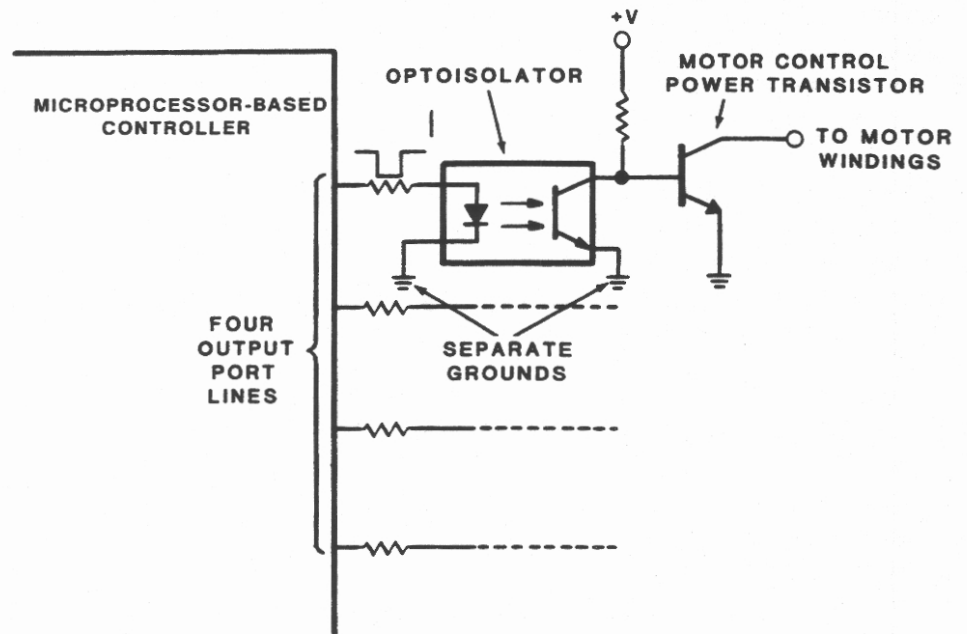


Figure 9-40

Increased isolation and protection is provided by using optoisolators in the motor/controller interface.

Once the interface is constructed, the stepper motor is controlled by simply writing the proper logic pattern to the output port. To advance the motor from a given position, the controller writes the next sequential bit pattern to the port. The motor direction is determined by the direction of the logic sequence. The motor speed is determined by how fast the controller writes the sequential bit patterns to the port.

Another type of stepper motor interface is illustrated in Figure 9-41. This interface uses a device called a **preset indexer**. Here, the controller writes a binary value to the port as before. However, the binary value represents the number of steps that the stepper motor is to be advanced. An additional port line is used to enable the preset indexer when the binary step value is written to the port. When enabled, the preset indexer automatically advances the stepper motor the number of steps specified by the controller. Less software is required for this interface. However, the trade-off is that more expensive hardware is required in the form of a preset indexer.

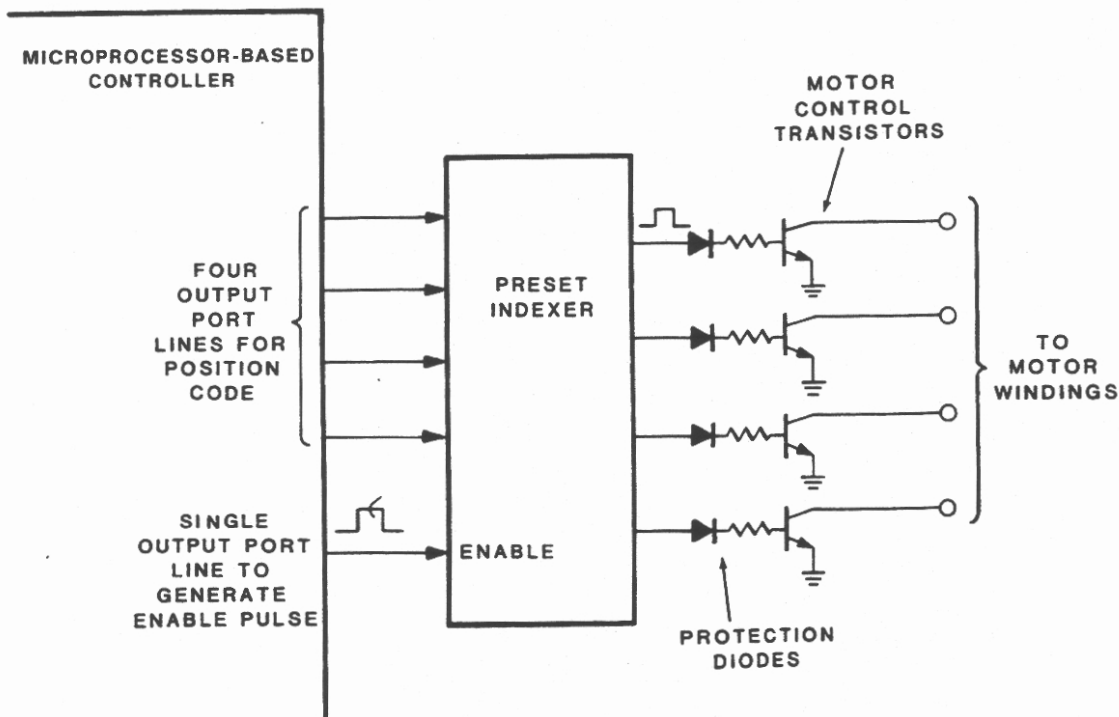


Figure 9-41

A preset indexer simplifies the motor control software and saves CPU time.

## Self-Test Review

16. List two ways to control a permanent magnet DC motor.

\_\_\_\_\_.

\_\_\_\_\_.

17. A 5 kHz TTL signal is high for .15 ms of its period.

A. What is the duty cycle of the signal?

B. What is the average DC voltage produced by the signal?

18. Explain the idea behind motor control using pulse-width modulation.

19. What is the major difference between the control circuits of a bipolar versus a unipolar stepper motor?

20. How can you improve the high speed performance of a unipolar stepper motor?

21. List the three transistor characteristics that must be considered when selecting control transistors for a stepper motor control circuit.

\_\_\_\_\_.

\_\_\_\_\_.

\_\_\_\_\_.

## Answers

16. Using a D/A converter.  
Using pulse-width modulation.

17. A. The period of a 5 kHz signal is 1/5 kHz, or .2 ms. Thus, the duty cycle is:

$$\begin{aligned}\text{Duty cycle} &= \frac{T_H}{\text{Period}} \times 100\% \\ &= \frac{.15 \text{ ms}}{.2 \text{ ms}} \times 100\% \\ &= .75 \times 100\% \\ &= 75\%\end{aligned}$$

- B. Using the above duty cycle the average DC voltage for the TTL signal is:

$$\begin{aligned}V_{DC} &= 5 \text{ V} \times \text{Duty Cycle} \\ &= 5 \text{ V} \times .75 \\ &= 3.75 \text{ V}\end{aligned}$$

18. A motor is controlled using pulse-width modulation by changing the duty cycle of a digital control signal. This changes the average voltage seen by the motor which, in turn, changes the speed of the motor via a power transistor arrangement.
19. A bipolar stepper motor control circuit requires twice as many control transistors as a unipolar control circuit.
20. You can improve the high speed performance of a unipolar stepper motor by adding a series resistance to each stator winding.
21. The transistor current rating.  
The transistor breakdown voltage.  
The transistor switching time.

## UNIT SUMMARY

Electronic control devices are often on/off devices that are used to activate the final control element in a process control loop. The digital signal cannot be used directly, since it does not have enough power to drive the final control element. Electronic control devices include: open collector drivers, bipolar and MOSFET power transistors, transistor arrays, peripheral power drivers, optocouplers, SCRs and TRIACs, to mention a few.

The final control element in an industrial process control loop is usually some sort of electro-mechanical device that converts the computer generated control signal into a mechanical action that has an effect on the process. These devices are referred to as actuators. Common actuators found in the industry include: valves, relays, solenoids, and motors. Probably the most common actuator found in the industry is the DC motor. Of particular importance are permanent magnet, wound-field, and stepper motors. Permanent magnet and stepper motors are especially suited to industrial applications since they are easily controlled from a microprocessor-based controller.

Permanent magnet DC motors are easily controlled using one of two techniques: with the output of a D/A converter or with pulse-width modulation. The speed of the motor depends on the average DC voltage level applied to its armature, while its direction of rotation depends on the polarity of the control signal.

Stepper motors are controlled by applying a digital code to the motor via several control transistors. The direction of rotation depends on the code sequence, while the speed of the motor depends on the rate at which the code sequence is applied. A bipolar control circuit is used to control a bipolar stepper, while a unipolar control circuit is used to control a bifilar, or unipolar, stepper. Three considerations must be taken into account when designing either a bipolar or unipolar control circuit. They are, the current rating, reverse breakdown voltage, and switching speed of the control transistors.

*Unit 10*

**MICROPROCESSOR APPLICATIONS**

## CONTENTS

|                                                  |       |
|--------------------------------------------------|-------|
| Introduction.....                                | 10-3  |
| Unit Objectives .....                            | 10-4  |
| Consumer Applications.....                       | 10-5  |
| The Automobile .....                             | 10-5  |
| Appliances.....                                  | 10-10 |
| A Weather Computer .....                         | 10-12 |
| Other Consumer Products That Think.....          | 10-15 |
| Personal Computers (PCs).....                    | 10-17 |
| Self-Test Review .....                           | 10-20 |
| Answers.....                                     | 10-21 |
| Industrial Applications .....                    | 10-22 |
| Production Industry Applications .....           | 10-22 |
| Robotics.....                                    | 10-22 |
| The Robot System .....                           | 10-23 |
| CAD/CAM .....                                    | 10-27 |
| Flexible Manufacturing Systems .....             | 10-29 |
| Commercial Aviation Applications.....            | 10-30 |
| Medical Applications .....                       | 10-33 |
| Self-Test Review .....                           | 10-36 |
| Answers.....                                     | 10-37 |
| Business Applications.....                       | 10-38 |
| Small Business Systems and Word Processors ..... | 10-38 |
| Copiers.....                                     | 10-40 |
| Cash Registers and Inventory Control.....        | 10-42 |
| Self-Test Review .....                           | 10-48 |
| Answers.....                                     | 10-49 |
| Unit Summary.....                                | 10-50 |

## ***Unit 10***

# **MICROPROCESSOR APPLICATIONS**

## **INTRODUCTION**

You have now been exposed to many of the basics required to use a microprocessor in real world applications. There is nothing magic in the design of a microprocessor applications circuit. You simply apply the basic sense and control principles presented in this course to the circuit design.

The microprocessor applications industry has taken two directions: dedicated applications for control, and system applications for data processing. A dedicated application is one in which the microprocessor is devoted solely to one major task, determined by the application. Dedicated applications include automobiles, typewriters, copy machines, industrial controls, appliances, toys, etc. Thus, dedicated applications are primarily sense and control applications. These make up over 50% of the total microprocessor applications market.

On the other hand, a systems application is one in which the microprocessor is user-controlled and programmable to perform a variety of computing tasks for the processing of data, or data processing. These applications include small business computers, personal computers, CAD/CAM, data acquisition systems, word processing systems, etc. Less than 50% of the total microprocessor applications market is systems oriented. However, the systems market is growing, and more systems applications are apparent, especially with newer generation high performance microprocessor devices.

In this unit, you will explore several real world microprocessor applications — both dedicated and systems. For clarity, we have divided the Unit into three applications categories: consumer, industrial, and business applications. Our intent is to present a few representative applications that should put things in perspective and give you a "flavor" for the possibilities that exist for applications of a microprocessor. In addition, we hope to stimulate your imagination to create new application ideas. You will find that there are many more applications than the ones discussed in this Unit. However, space and time do not permit a discussion of every possible microprocessor application. Furthermore, new microprocessor applications circuits are being created daily. Actually, the only limiting factor to microprocessor applications is the imagination.



## UNIT OBJECTIVES

When you have completed this unit, you should be able to:

1. Describe the two marketing directions that have been taken by the microprocessor applications industry.
2. Explain how a microprocessor is used to control exhaust emissions and fuel economy in an automobile.
3. State the features of a microprocessor-based weather station.
4. List several consumer product applications of a microprocessor.
5. Explain how multiple microprocessors are utilized in advanced personal computer and business systems.
6. List the three major sections of an intelligent robot, and the three major categories of industrial robots.
7. Define sensory feedback, CAD/CAM, and FMS.
8. Describe a typical CAD/CAM system.
9. List at least three advantages of using a CAD/CAM system in a manufacturing process.
10. Describe a typical flexible manufacturing system, or FMS.
11. Suggest several microprocessor applications in the aviation and medical industries.
12. Explain several business applications of microprocessors, including small business computers, word processors, copiers, cash registers, and inventory control systems.

## CONSUMER APPLICATIONS

This category is potentially the largest microprocessor applications market. It started with the microwave oven and small pocket calculators, and has exploded to include automobiles, toys, games, appliances, cameras, and personal computers, just to mention a few. Now, let's take a closer look at a few representative consumer applications of a microprocessor.

### The Automobile

It is predicted that the automobile will be the largest single application of the microprocessor. Most automobiles produced today already have an "on board" microprocessor circuit that is used to control a variety of functions.

The need for some sort of microprocessor control in an automobile became apparent with increased emission control requirements and the need for better fuel economy. With microprocessor control, the automobile is not only cleaner and more fuel efficient, but its performance is also enhanced. For this application, a microprocessor control circuit like the one shown in Figure 10-1 is used to sense various engine conditions, and control the air/fuel mixture to the carburetor and the engine timing.

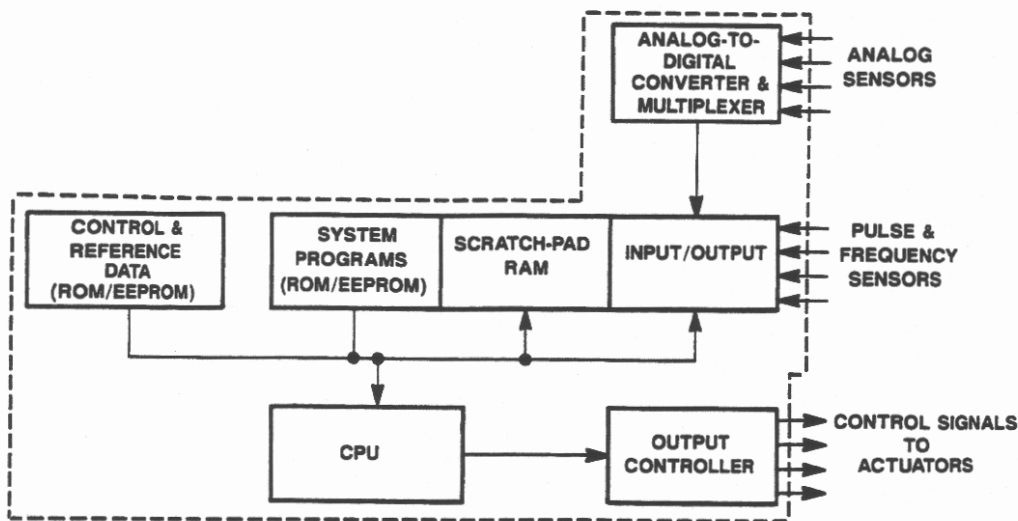


Figure 10-1

Block diagram of a microprocessor engine control system.

To control air/fuel mixture and engine timing, a microprocessor must sense several dynamic parameters within the automobile engine. Position sensors must be used to determine the position of the crankshaft, throttle, and exhaust valves. Temperature sensors are used to measure coolant and inlet-air temperature. Pressure sensors are used to determine manifold pressure and external barometric pressure. As the engine is operated, the microprocessor accumulates data from the engine sensors and makes a control decision to adjust the air/fuel mixture to the carburetor. In addition, the engine timing is also controlled by the microprocessor. Control decisions are made up to 30 times a second to provide continuous "fine tuning" of the engine's performance.

To make the control decision, the microprocessor compares the sensed data to a data look-up table located in ROM. The ROM look-up table contains optimum performance data obtained from numerous lab tests and pre-production test cars. Most automobile systems today use mask programmed ROMs. However, in the future, special programmed ROMs, such as EEPROMs, or Electrically Erasable Read-Only Memory, will be used to store the control data. EEPROMs will allow a microprocessor in an automobile to modify the control data look-up tables according to engine wear. This will provide optimum engine performance throughout the life of the automobile. As you can see from the block diagram in Figure 10-1, most of the features required for an automobile application are internal to the microprocessor chip. In fact, *single-chip microcomputers* have been developed specifically for the automobile industry, and are currently used by both Chrysler and A/C Delco, among others.

A microprocessor's time would be wasted if it were only used for emission control and fuel economy. These tasks actually require less than 10% of a standard 8-bit microprocessor's time. As a result, the microprocessor is available to perform an endless variety of additional tasks, some of which are illustrated in Figure 10-2. For example, the microprocessor can be used for security, climate control, cruise control, trip computing, dash display, and engine diagnostics, just to mention a few. The list is almost endless, since a microcomputer system is dedicated to a single automobile. As the years go by, you are sure to see more and more applications of a microprocessor in the automobile.

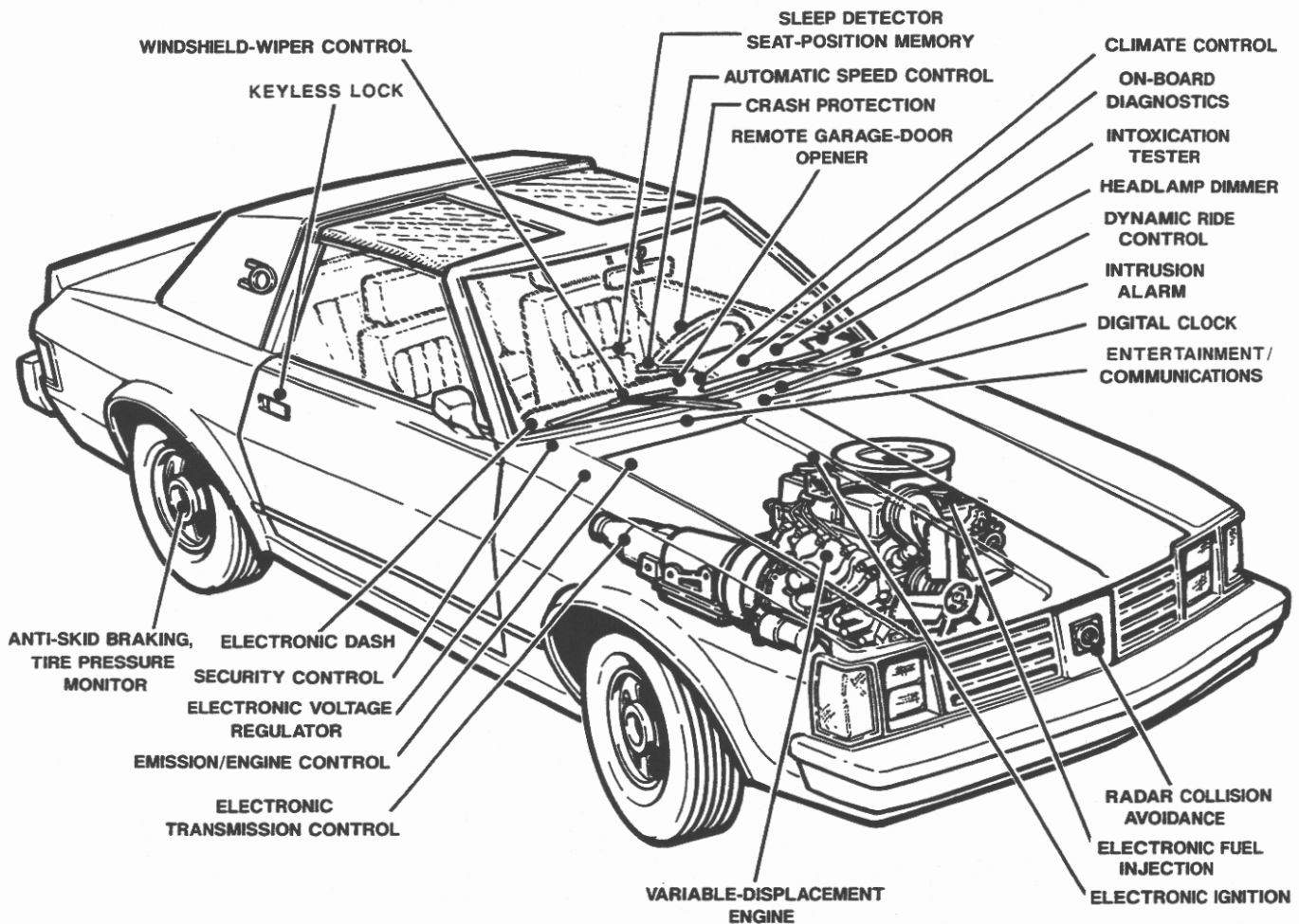


Figure 10-2

Possible applications of a microprocessor in an automobile.

The electronic traction control system shown in Figure 10-3 is another interesting automotive application of a microprocessor. This system has already been adopted by a manufacturer of a high performance turbo-charged automobile. Electronic traction control is the opposite of anti-skid braking. Anti-skid braking prevents wheel lockup during braking, while electronic traction control prevents wheel spinning during acceleration.

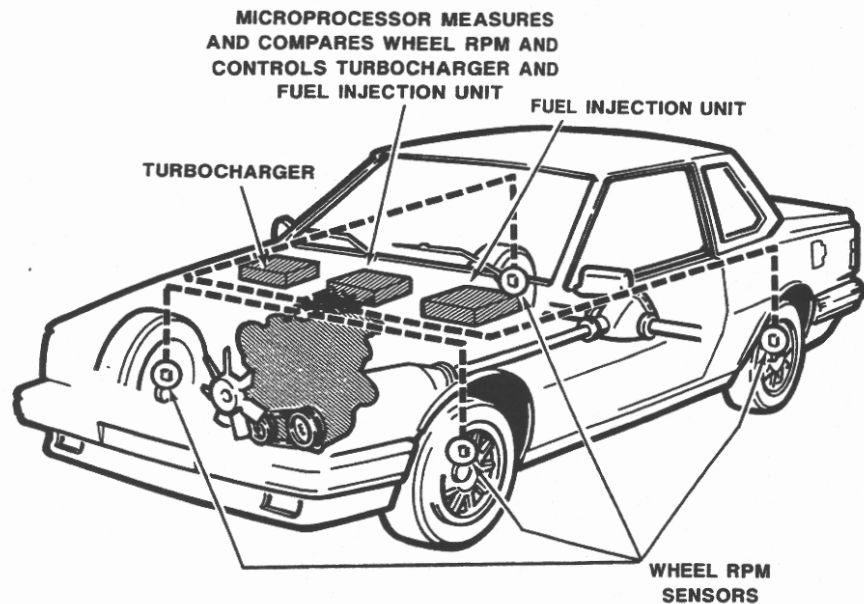


Figure 10-3

An electronic traction control system uses a microprocessor to detect relative wheel RPM and control the engine performance.

Here's how it works. The microprocessor senses the rotational speed of each wheel. This is accomplished with magnetic sensing devices such as those described in Unit 8. The output of each magnetic sensor is a series of pulses whose frequency is proportional to the rotational speed of the wheel. The microprocessor compares the speed of each rear wheel to that of the free-rolling front wheel on the same side of the automobile by comparing the frequency output of the respective wheel sensors. When the front wheel speed on either side differs from the rear wheel speed on the same side by a given percentage, the microprocessor reduces the engine power.

Engine power reduction is accomplished in two ways. First, a portion of the output of the engine turbo-charger is diverted from the engine. If this does not stop the wheelspin, the microprocessor begins reducing the amount of fuel injected into the engine cylinders. The fuel injection is reduced in stages, first affecting one cylinder, then two, then three, and so on, until the wheelspin is stopped and the automobile regains traction. When traction is regained, the engine power is progressively restored by reversing the above power reduction sequence.

Diagnostics of engine problems is another important application of a microprocessor in the automobile industry. Some systems allow the operator or service technician to execute a diagnostic program that will check all the sensors in the system. Any malfunction is displayed on a dashboard panel or a portable engine analyzer. Other systems provide continuous monitoring of the engine sensors and engine conditions. When a problem is detected, a warning light is flashed to the operator. At the same time, the microprocessor stores data about the problem in a nonvolatile memory. The operator must then take the automobile to a qualified service center. The service center technician executes a service routine that reads the system memory. When the service routine is executed, the microprocessor generates coded messages that indicate the precise nature of the problem.

One automobile manufacturer is developing a small transmitter that will be installed with the microprocessor system in its future automobile production. The use of such a system is illustrated in Figure 10-4. When you buy a new car, the dealer will install a small inexpensive radio transmitter. When the automobile is brought in for service, the transmitter transmits a coded vehicle I.D. number and any self diagnostic information that has been accumulated in the automobile system's memory. A computer at the entrance of the service center receives and analyzes the transmitted data and generates a printout for the service technician. The printout contains the past servicing history of the automobile as well as any self diagnostic information. In effect, the computer printout is a work order to be followed by the service technician. With self diagnostics, the service technician does not have to be an electronics engineer to diagnose problems and repair the system.

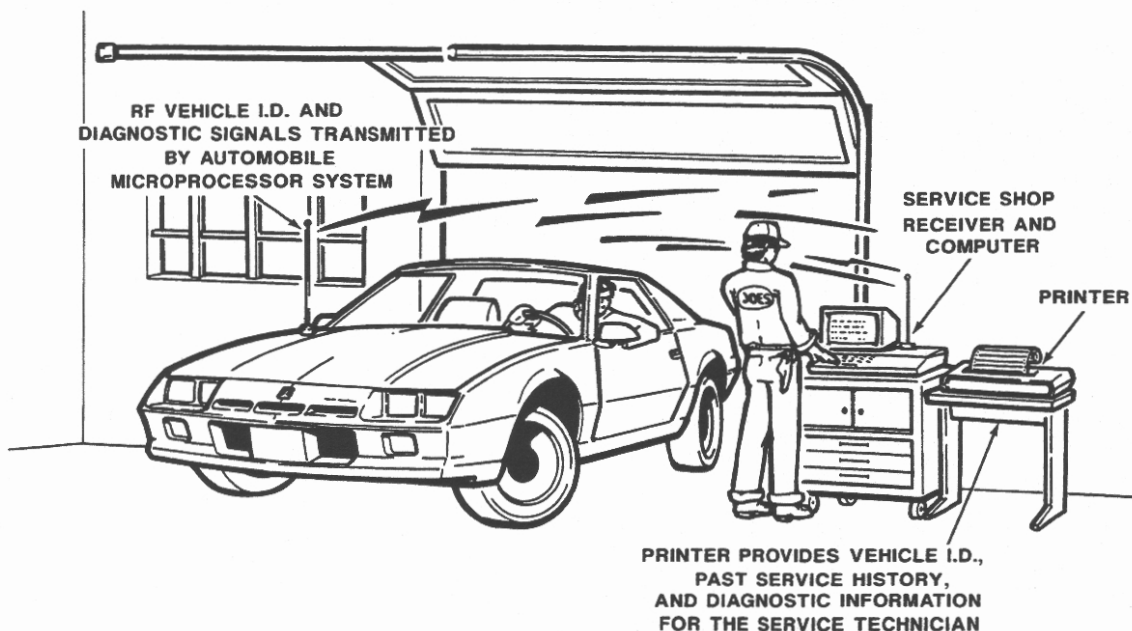


Figure 10-4

An automated automobile service center.

## Appliances

The very first high volume consumer application of a microprocessor in a home appliance was in a microwave oven. It is, therefore, only fitting that a microwave oven is used to illustrate the application of a microprocessor in an appliance.

A typical microprocessor control system for a microwave oven is shown in Figure 10-5. At the heart of the system is a microprocessor, or MPU, along with a small amount of scratch-pad RAM and enough ROM to store the system programs. Many times, the MPU, RAM, and ROM are all integrated into a single-chip microcomputer. This allows for fewer system components.

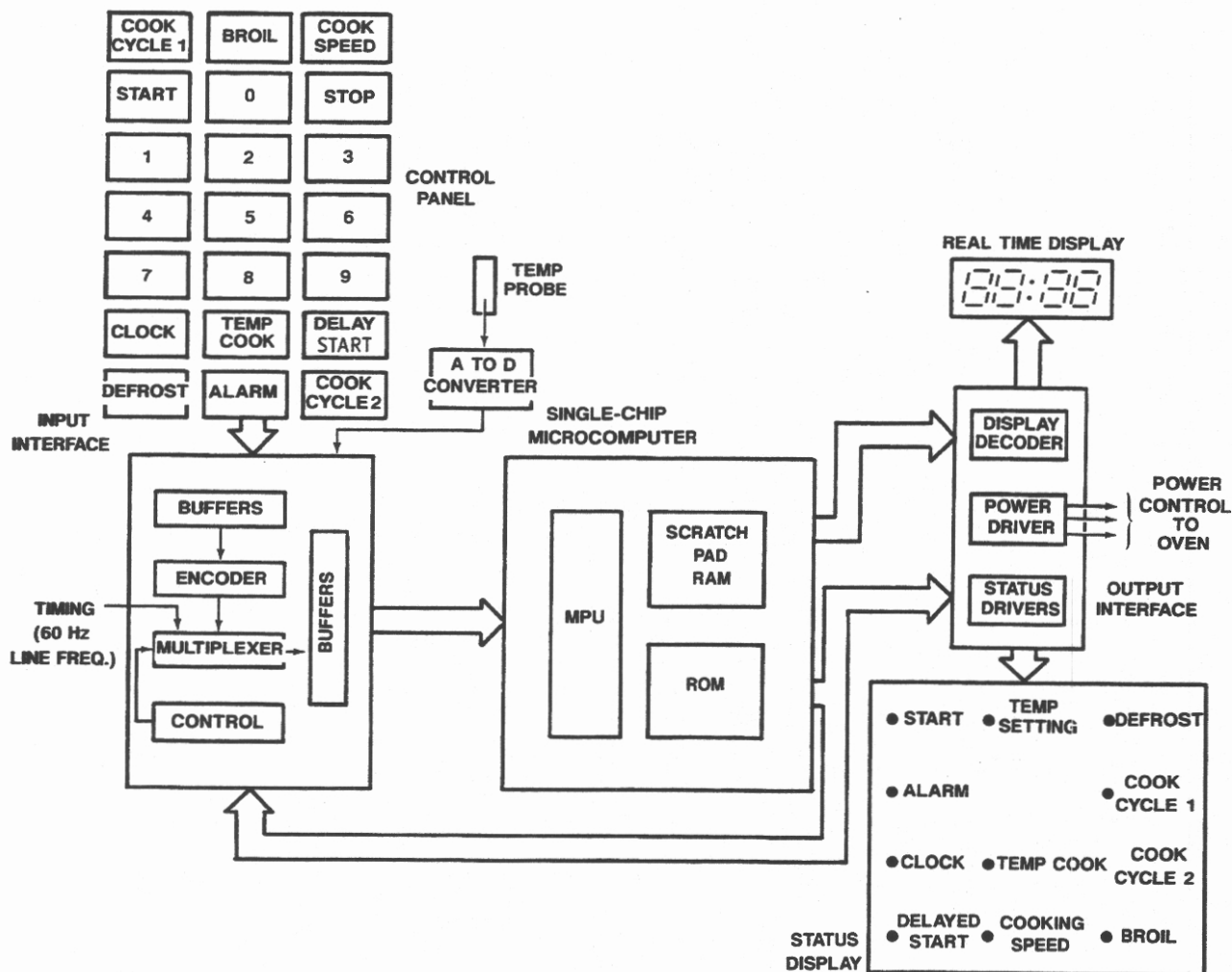


Figure 10-5

A typical microprocessor control circuit for a microwave oven.

Aside from the microcomputer section, the system consists of an input section and an output section. The user enters various cooking information via the control panel keypad. Single key features which are available include several cooking cycles, a broiling cycle, and a defrost cycle. In addition, the control panel is used to program a delayed oven start and to set the time-of-day clock. The information input via the control panel is stored in RAM and used by the MPU to execute the required cooking cycles. All of the system software is contained in ROM. The ROM software is basically a series of time delay programs that are used to control the various cooking cycles along with the time-of-day clock.

During a given cooking cycle, temperature information is obtained via a thermistor type temperature probe. The probe output is applied to an A/D converter for conversion to a 4-bit digital value that is used by the MPU to make temperature control decisions.

The output section of the system consists of a time-of-day digital clock display and an LED status display. The time-of-day clock is set via the control panel and is controlled by the MPU using a "real time" ROM program. The status display panel provides the user with a visual indication of the cycle that is in progress.

Notice from Figure 10-5 that the output section also includes a power driver. The power driver receives control information from the microcomputer section. The control information is amplified and passed on to the microwave element in the oven.



## A Weather Computer

A personal microprocessor-based weather station is pictured in Figure 10-6. This unit displays time/date, indoor and outdoor ( $^{\circ}\text{F}$  or  $^{\circ}\text{C}$ ) temperatures, wind speed, wind direction, and barometric pressure. The weather computer not only monitors current information, but also reads significant weather changes and updates its memory accordingly. From stored information, the computer calculates high/low temperatures and barometric pressures, current rate and direction of barometric change, average wind speed, and wind chill factor. This allows both present and past weather data to be displayed, and assists an individual in predicting future weather patterns.

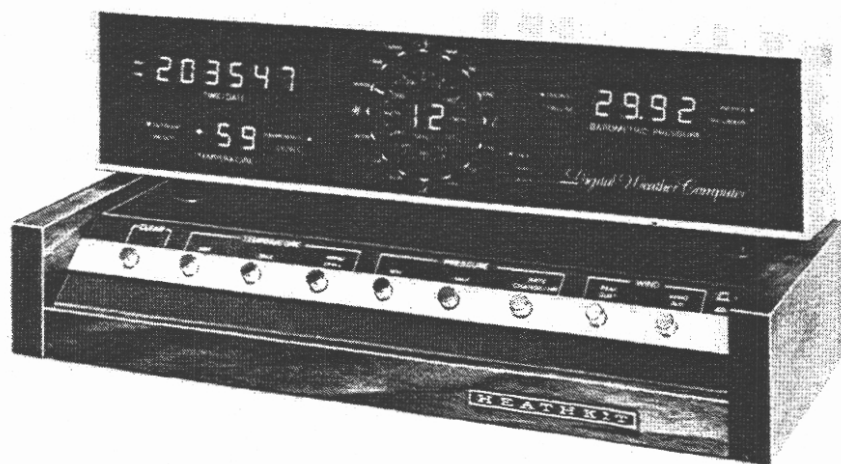


Figure 10-6  
The Heath/Zenith ID4001 weather computer.

A block diagram of the weather computer circuit is shown in Figure 10-7. The brains of the system is a microprocessor which receives input signals from several temperature, pressure, and wind sensors/transducers. The microprocessor processes these signals for use by the display and memory circuits.

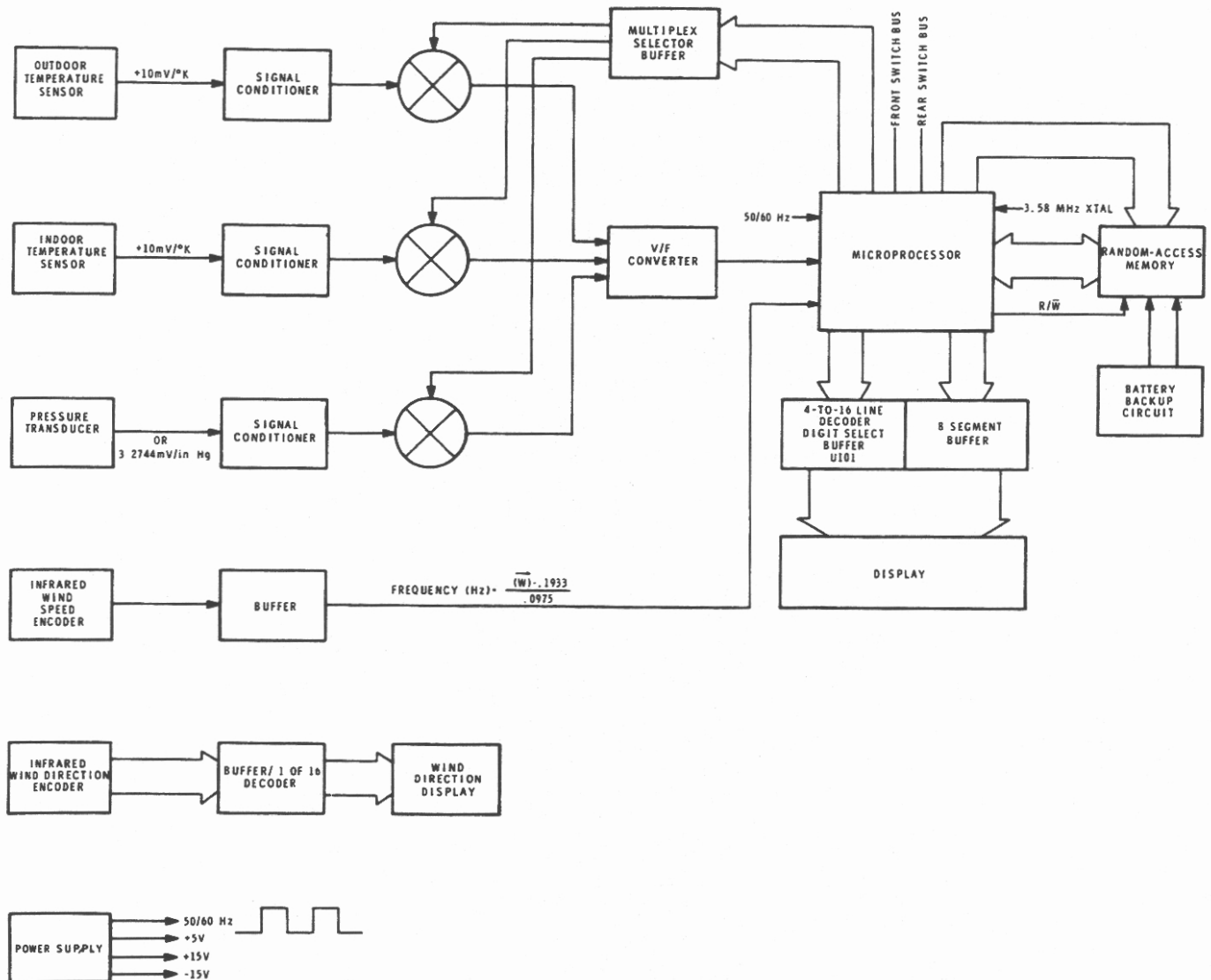


Figure 10-7  
Block Diagram of a microprocessor-based weather computer.

The outdoor and indoor temperature sensors are semiconductor type sensors that generate a voltage ( $10 \text{ mV}/^\circ\text{K}$ ) that is linearly proportional to temperature. The generated voltage is amplified by a signal conditioner and multiplexed into a V/F converter. The V/F converter generates a frequency that is proportional to its input voltage and applies this frequency to the microprocessor circuit where it is measured and stored.

The pressure transducer is a piezoresistive type device that produces a voltage which is proportional to barometric pressure. Like the temperature signals, the pressure signal is amplified by a signal conditioner and multiplexed into the V/F converter to generate a frequency proportional to the barometric pressure.

Wind information is sensed by two sensors: an infrared speed encoder and an infrared wind direction encoder. The wind speed sensor generates pulses that are proportional to the rotational speed of an external anemometer. The pulses are generated using an optical technique as discussed in Unit 7. The wind speed pulses are buffered and applied directly to an interrupt input of the microprocessor for measurement, display, and storage. The wind direction sensor is also an optical device that produces a 4-bit binary Gray code to indicate the angular position of an external weather vane. The 4-bit Gray code is buffered, decoded, and applied directly to a circular pattern of 16 LED wind direction indicators on the display console. Only one of the 16 LED indicators will illuminate to show the wind direction.

In addition to displaying current and past weather data, the weather computer can also be connected to an external chart recorder. A chart recorder can be used to provide a graphical display of changing weather information.

## Other Consumer Products That Think

One of the first applications of a microprocessor was in general purpose electronic calculators. These general purpose calculators have evolved into a variety of special purpose products for teachers, business people, athletes, and just about anyone.

One of the first calculator type products was the Speak & Spell™ shown in Figure 10-8. Speak & Spell™ uses a microprocessor-controlled voice synthesizer to ask you to spell a word. As you enter the letters of the word, the unit pronounces the entered letters. After the word has been entered, Speak & Spell™ "tells" you if the spelling is correct. If not correct, you get one more chance. If wrong again, the unit spells the word correctly for you.

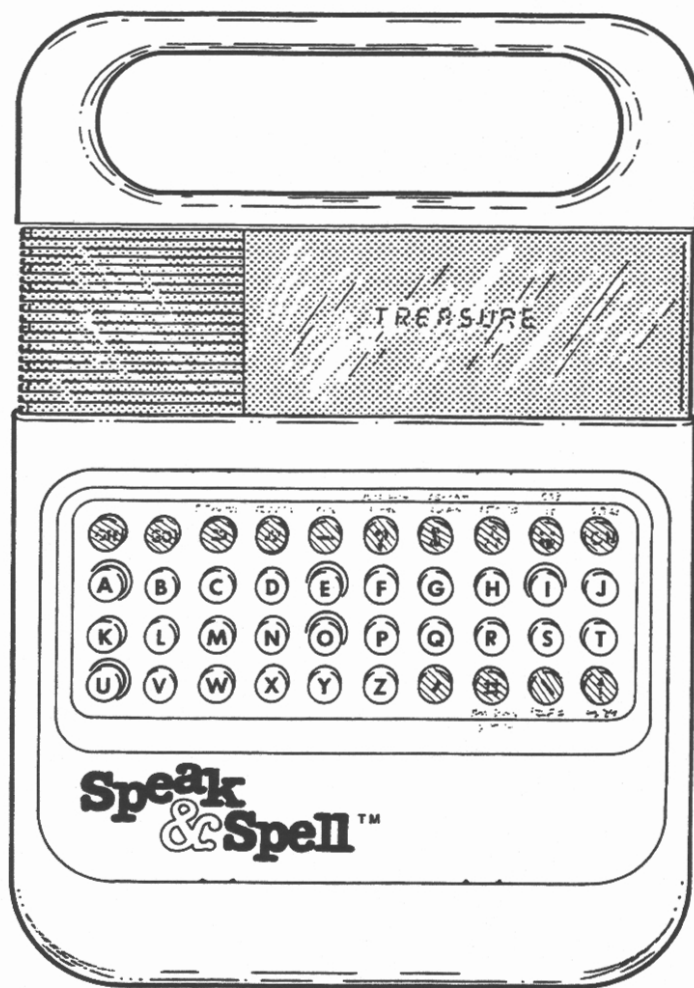


Figure 10-8

Speak & Spell™: a microprocessor-based teaching machine.

Several other consumer products that think are pictured in Figure 10-9. Products such as these are just the beginning. Future microprocessor-based consumer products will only be limited by the imagination.

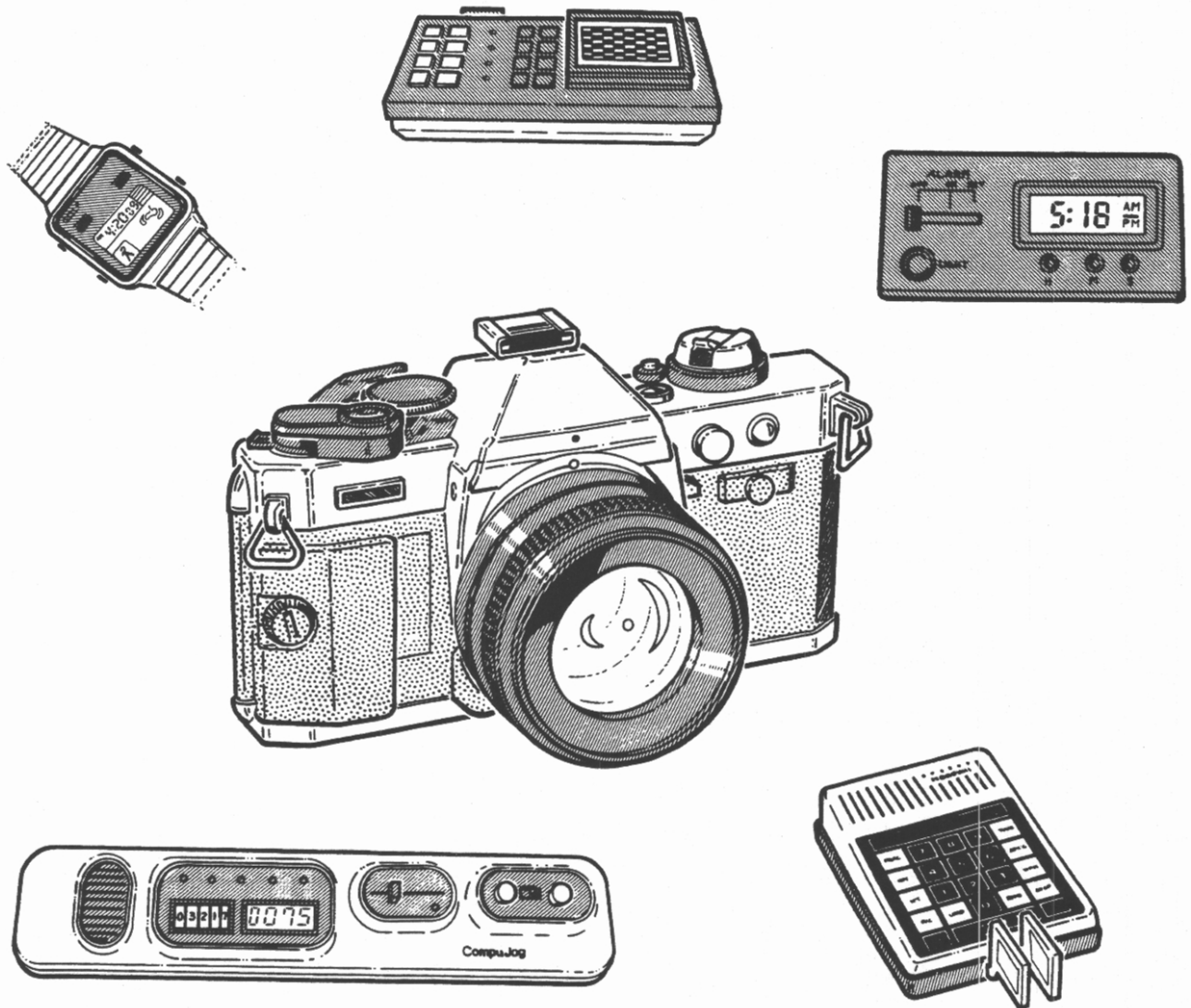


Figure 10-9  
Other consumer products that think.

## Personal Computers (PCs)

Personal computers, like those pictured in Figure 10-10, have become commonplace in today's society. What started out as a hobbyist item for engineers, technicians, and programmers, has now become almost as common in society as the TV set. The market has exploded to include not only the hobbyist market, but the general purpose home, business, and educational markets as well. Moreover, the onslaught of personal computers has created a need for computer literacy within our society, in addition to the "3 R's" of reading, writing, and arithmetic. Many students are now required to take fundamental computer literacy courses in elementary school.

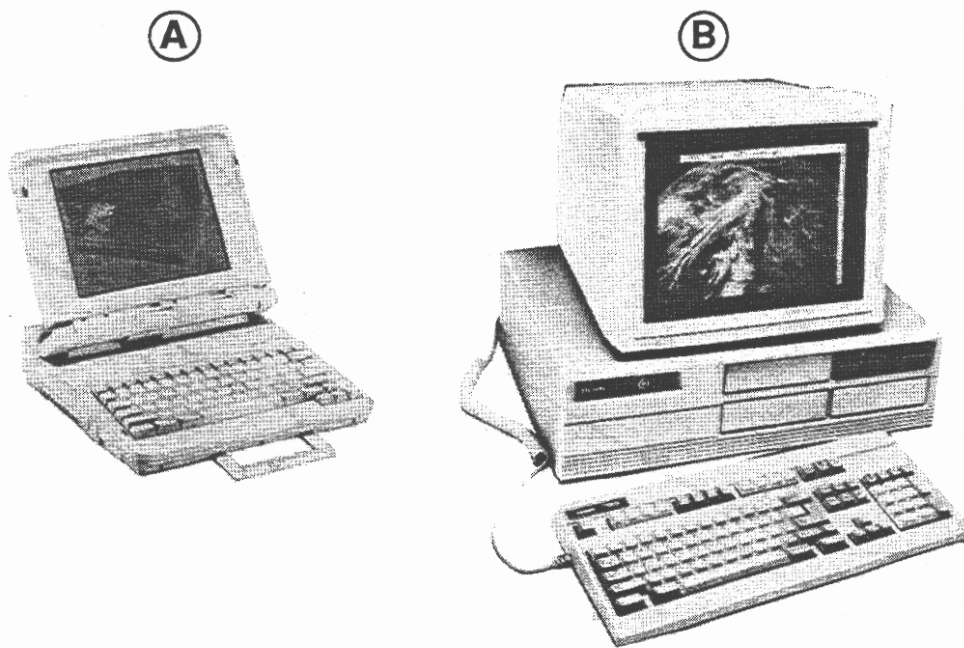


Figure 10-10

Typical personal computers, the Heath/Zenith A. HS-2860 and B. The H-386 (Shown with ZCM-1490 Flat-Tension Monitor)

The predicted use of personal computers in the United States is illustrated by the graph in Figure 10-11. This graph shows the predicted total number of personal computers in use in the home, business, and education extrapolated through the year 1992. What has created such a demand for these units is their decreased cost and increased computing capabilities. A typical 16-bit microcomputer system today can perform many of the computing tasks that required a large and expensive mainframe computer in the 1960's. As a result, the largest market for personal computers is the business market. However, because they are so inexpensive, tremendous computing power can also be brought into the home and classrooms.

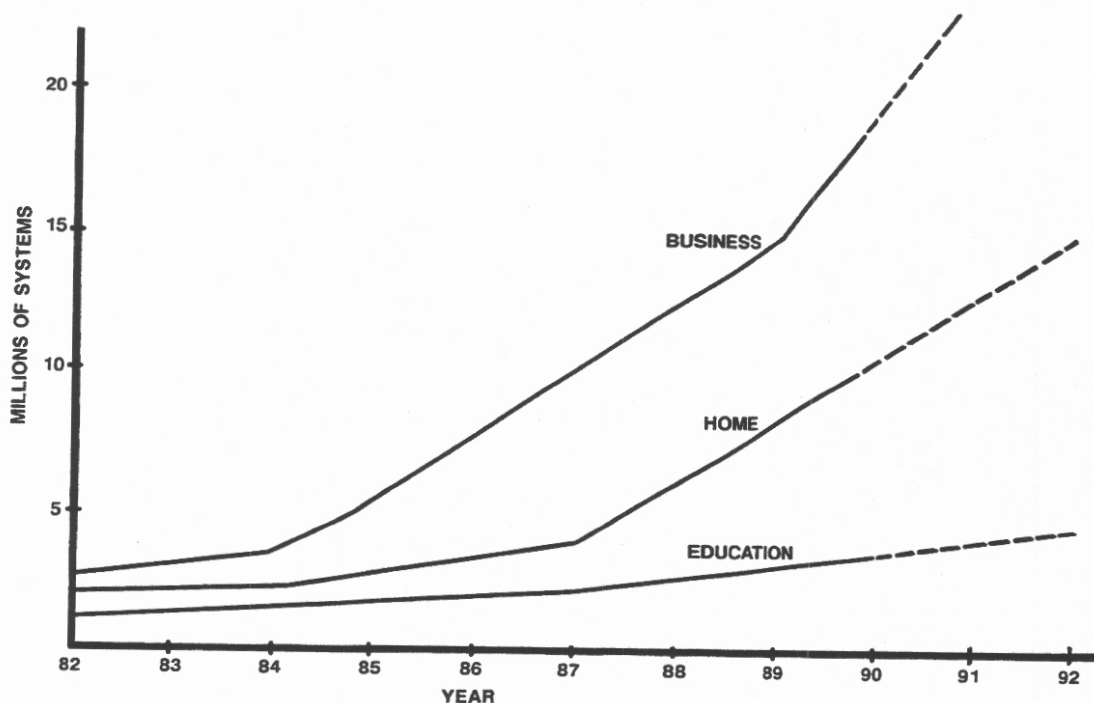


Figure 10-11

The total predicted number of microcomputer systems installed in the United States through the year 1992.

Personal computers have evolved from single 8-bit microprocessor machines to 16-bit multi-microprocessor systems. The block diagram in Figure 10-12 illustrates how multiple microprocessors can be used in a 16-bit system. A 16-bit microprocessor handles all the data processing operations and supervises the operation of the entire system. One or more 8-bit microprocessors are dedicated to handling the system input and output, or I/O, operations. This frees the 16-bit microprocessor from the I/O tasks. As a result, data can be processed by the 16-bit microprocessor simultaneously while I/O is being handled by the 8-bit microprocessor(s). In addition, programs that were written for earlier 8-bit systems can be run on the 16-bit system. Many manufacturers, including Zenith Data Systems, have taken this design approach in their 16-bit systems.

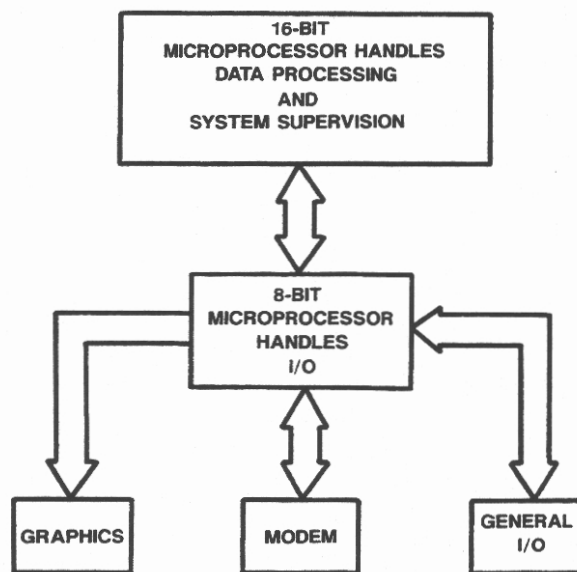


Figure 10-12

A 16-bit multi-microprocessor personal computer system.



## Self-Test Review

1. What are the two major marketing directions that have been taken by the microprocessor applications industry? \_\_\_\_\_  
\_\_\_\_\_
2. What is predicted to be the largest single application of a microprocessor. \_\_\_\_\_  
\_\_\_\_\_
3. Explain how a microprocessor is used to control exhaust emissions and fuel economy in an automobile. \_\_\_\_\_  
\_\_\_\_\_
4. List at least five sense/control tasks that a microprocessor can perform in an automobile besides exhaust emission and fuel economy control. \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_
5. Why is a microprocessor in an automobile important to a service technician? \_\_\_\_\_  
\_\_\_\_\_
6. State the features of a microprocessor-based weather station. \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_
7. List at least five additional consumer product applications of a microprocessor. \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_
8. Explain how multiple microprocessors are used in a 16-bit personal computer system. \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

## Answers

1. The two major marketing directions taken by the microprocessor applications industry are dedicated applications for control and systems applications for data processing.
2. The automobile is predicted to be the largest single application of the microprocessor.
3. A microprocessor in an automobile controls exhaust emissions and fuel economy by controlling the air/fuel mixture and the engine timing.
4. Sense and control applications that can be performed by a microprocessor in an automobile include: security, dashboard display, climate control, entertainment/communications, collision avoidance, anti-skid braking, trip computing, cruise control, and engine diagnostics. See Figure 10-2 for additional applications.
5. A microprocessor in an automobile is important to a service technician because of its diagnostic capabilities.
6. The features of a microprocessor-based weather computer include: display of time/date, indoor and outdoor temperature, wind direction, and barometric pressure. In addition, the weather computer can read significant changes in weather and calculate weather extremes to aid in weather forecasting.
7. Additional consumer product applications of a microprocessor include cameras, appliances, watches, calculators, games, toys, sports, and personal computers, just to mention a few.
8. In a 16-bit multi-processor personal computer system, a 16-bit microprocessor handles all the data processing operations and supervises the operation of the entire system. One or more 8-bit microprocessors are dedicated to handling the system I/O operations. In addition, the 8-bit microprocessor(s) can be used to execute 8-bit programs on the 16-bit system.

## INDUSTRIAL APPLICATIONS

Many say that we are presently experiencing a second industrial revolution as a result of microprocessor technology. Recall that the first industrial revolution took place at the turn of the last century when machines began replacing muscle. This same type of revolution is taking place now, except that machines are not only replacing muscle, but also minds.

In this section, you will examine several applications of microprocessors in industry. You are already familiar with how a microprocessor is used in an industrial process to sense and control an operation. Now, you are ready to go one step further and explore the fields of robotics, CAD/CAM, and flexible manufacturing systems. In addition to production industry applications, you will also see how microprocessors are being applied in other industries such as the aviation and medical industries.

### Production Industry Applications

With the advent of the microprocessor, a computer intelligence can be dedicated to simple and complex industrial operations. In most cases, the result is increased productivity, quality, and safety.

Industrial robots which use microprocessor controls now perform such routine tasks as welding, spray painting, and assembly, which normally pose health hazards to us humans. The field of robotics is just in its infant stages. Future robots will handle most of the routine and hazardous tasks in a production process — freeing the production worker to perform more meaningful tasks.

Microprocessor-based systems are also being used to design products, and control the entire manufacturing process. This technology is referred to as computer aided design and computer aided manufacturing, or CAD/CAM. The combination of robotics and CAD/CAM results in a flexible manufacturing system, or FMS. Now let's take a closer look at robotics, CAD/CAM, and FMS.

### ROBOTICS

Robotics is a direct application of microprocessor technology. In fact, the field of robotics is probably the biggest application challenge for the microprocessor and its related sense and control circuitry. All of the sense, control, and decision making principles that you have learned up to this point must be used to produce an "intelligent" robot.

Any intelligent robot can be broken down into three major sections: **power supply, controller, and manipulator**. The power supply supplies electrical, hydraulic, and/or pneumatic energy to the robot. Some robots only require electrical energy to operate the controller circuit and a series of motors for movement. Other robots, especially industrial robots, require hydraulic and/or pneumatic (air) energy, in addition to electrical energy, to operate.

The controller is the brains of the robot. Today's robots use a microprocessor-based controller that is programmed to sequence the robot through a series of desired tasks. In addition, more advanced robots use the microprocessor and related sensing circuits to sense external conditions and make "in course" correction decisions while sequencing through a given task. The intelligence of the robot is directly related to the programming capabilities and the sensors of the microprocessor-based controller.

The manipulator is the final control element that allows the robot to perform work. The robot manipulator usually consists of a mechanical arm, wrist, and hand, or gripper, assembly. The operational flexibility of a robot is related directly to the sophistication of its manipulator. Simple robots only provide one or two axes of motion for the manipulator, while advanced robots provide up to ten.

## The Robot System

When you think about it, a computer controlled robot is nothing more than a closed loop process control system. The entire operation of the robot reduces to the three closed loop control tasks of **sensing, decision-making, and control**. This idea is illustrated by the robot system in Figure 10-13. Here, a single robot is performing a single task. Such a system is called a **robot workstation**.

Let's take a closer look at the task at hand. The task is a bin-picking operation that may look simple, but is very difficult for a robot. Notice that there are three parts: a disc, a cylinder, and a ball. There is a separate bin for each part. The robot must pick-and-place a disc, cylinder, and ball, and place them on the conveyor in that order for a subsequent welding operation. Assuming the parts have been randomly dumped into their respective bins, the robot must first locate a given part, determine its orientation, pick the part up without dropping it, and place it on the conveyor.

The first difficult sensing task is to identify a given part and determine its orientation so that it can be grasped by the robot. In Figure 10-13, a vision system is being used within the workstation to identify the parts and determine their orientation. The sensory feedback provided by the vision system allows the robot to manipulate to the part and orient its gripper to grasp the part.

Next, the robot must lift the part out of its bin. Again this might sound like a simple task, but could present some difficulty unless the robot is equipped with **tactile sensing**. The robot must determine when the part has been grasped properly using its tactile sensing ability. When it begins to lift the part out of the bin, the robot must use its **slip sensing** ability to adjust the gripping force. This would be especially critical if the weight of the parts varied significantly.

Finally, once a part is grasped, the robot must be programmed to place the part on the conveyor, go to the next part bin, and repeat the bin-picking operation.

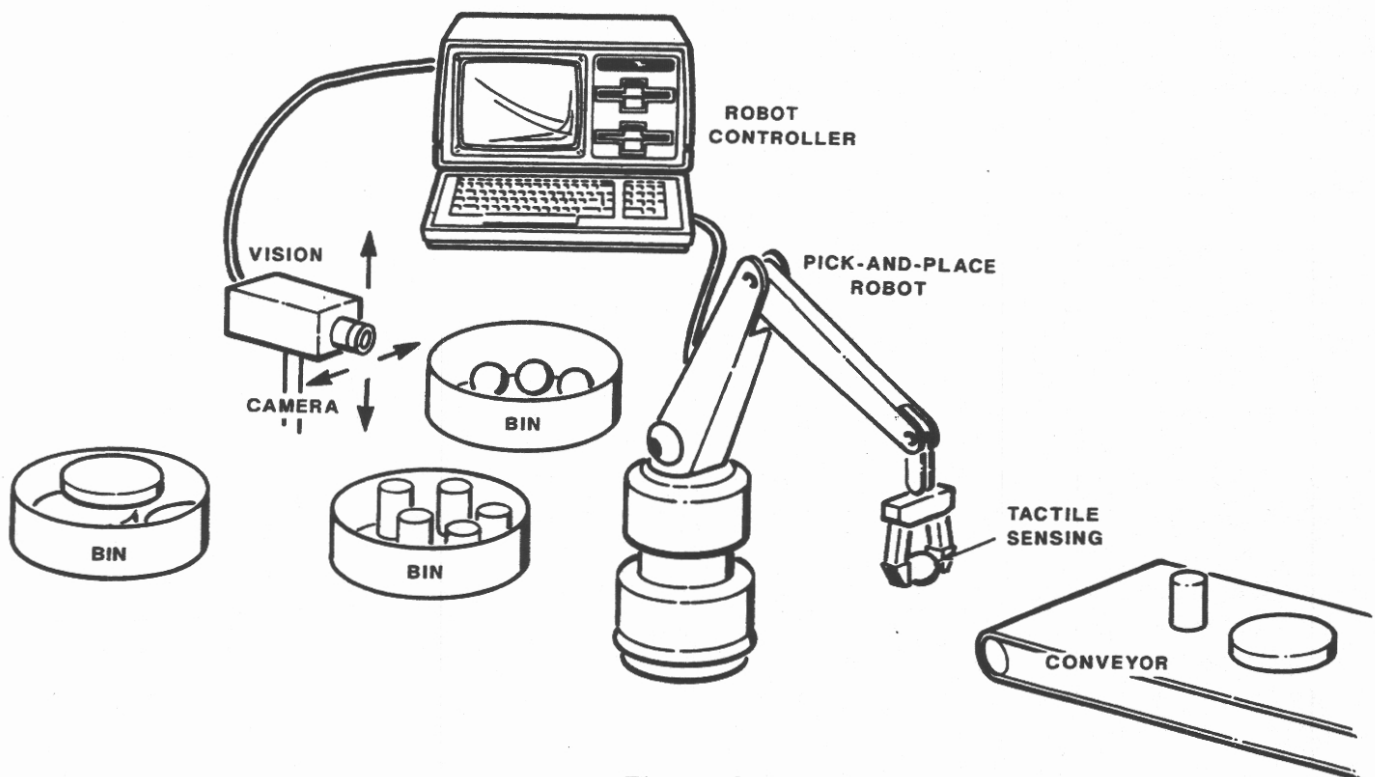


Figure 10-13  
A robot workstation.

The process has been extended in Figure 10-14. As you can see, a second robot has been added to perform the part assembly operation. The task at hand here is to assemble the parts as they move down the conveyor. Notice that an optical scanning system is used to count the parts, and "tell" the assembly robot when a part is present. Unless a special holding fixture is used, the assembly robot must also be equipped with vision and tactile sensing for the assembly operation. Such sensing abilities assure that the parts are placed together to precise specifications.

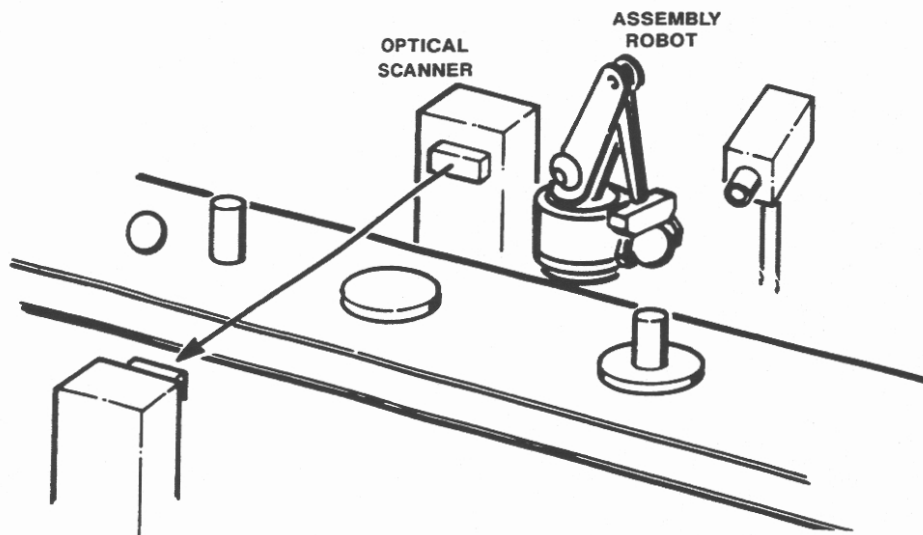


Figure 10-14

An assembly robot working with a pick-and-place robot.

Now look at Figure 10-15. In this figure the process has been extended by adding a third robot (a welding robot) to join and assemble the parts together. Again, a scanning system is used to count the parts and trigger the welding robot into action. This robot might also be equipped with a vision and/or tactile sensor to guide the robot along the weld seam. In addition, a vision system could serve a dual purpose here by inspecting the weld seam as it is formed. When any pits or voids are detected, the robot would be programmed to re-weld the seam until an acceptable weld is produced.

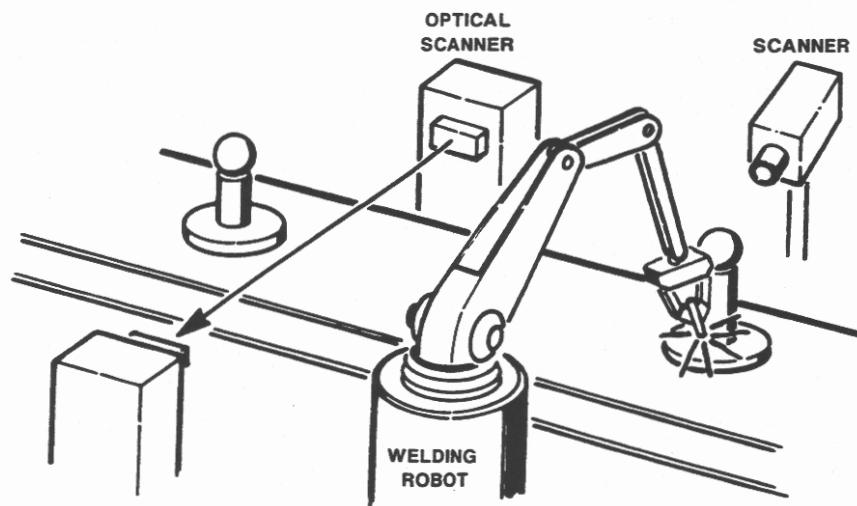


Figure 10-15  
A welding robot welds the assembly.

Finally, a fourth robot has been added in Figure 10-16 to pick the finished parts from the conveyor and place them onto a carrier for transportation to a storage area. In addition, a vision system has been added to provide a 100% final inspection of the finished product. The vision system would inspect the assembled parts for proper orientation and alignment, as well as for cosmetic defects. The system would direct the final pick-and-place robot to place acceptable parts onto the transportation carrier and reject parts into a rejection bin for engineering disposition.\*

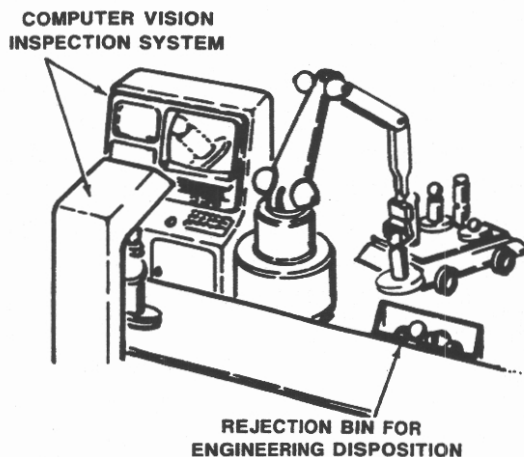


Figure 10-16  
A computer vision system performs the final product inspection,  
and directs a pick-and-place robot.

\* For further information see *Industrial Robotics and Automation*, Heathkit/Zenith Educational Systems.

## CAD/CAM

Now, let's go one step further with our process and add CAD/CAM to the system as shown in Figure 10-17. What is CAD/CAM? Literally, the acronym CAD/CAM stands for **Computer-Aided Design/Computer-Aided Manufacture**. In other words, CAD/CAM is the technology of using computers in the production process, from early product design and development to final assembly and testing.

CAD/CAM is presently having a major impact on design engineering, manufacturing, and documentation in many industries. By using CAD/CAM, a manufacturer can increase productivity, decrease costs, increase product quality, and improve his/her competitive position in the marketplace. Experience has shown that CAD/CAM increases productivity five-fold, with some manufacturers experiencing as much as a twenty-fold increase in productivity.

Using a keyboard, graphic tablet, and light pen, a product designer can create, modify, and refine a product design while viewing his/her work on a graphics display terminal. With simple input commands, the designer can magnify, rotate, turnover, copy, or change any or all of the product design graphics. Moreover, the designer can add depth to produce a 3-D image of the product. Finally, the system will generate front, top, and side views with associated dimensions and text to produce the finished product drawing.

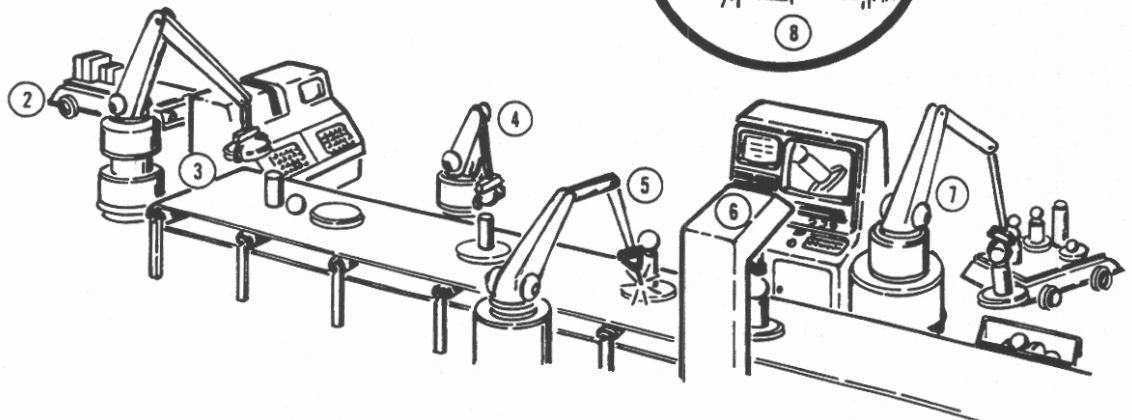
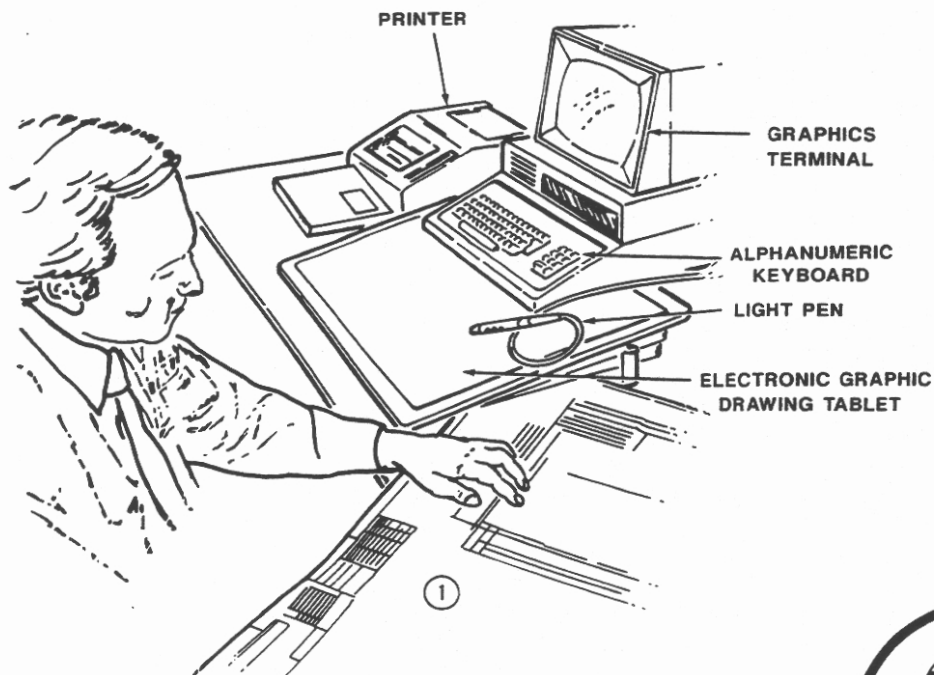
Once the product design is completed, it is stored in mass memory. As a result, design changes are simple. To aid in manufacturing, the CAD/CAM system is used to design tooling and fixturing, generate NC (numerical control) machine tapes, and to actually program assembly robots and automated test equipment. In fact, a CAD/CAM system can even simulate the production of a part with software prior to any actual production operations.

Combined with robotics, CAD/CAM\* can be used to create completely computerized and automated manufacturing processes. Such processes are called **flexible manufacturing systems** — the topic of the next discussion.

---

\* For further information see *CAD/CAM: A Hands On Approach*, Heathkit/Zenith Educational Systems.





**Figure 10-17**  
A typical flexible manufacturing system, or FMS.

## FLEXIBLE MANUFACTURING SYSTEMS

A flexible manufacturing system, or FMS, incorporates CAD/CAM and robotics to produce the ultimate automated manufacturing process. The process that we have been building up to in Figure 10-17 represents a typical FMS. Let's summarize the entire process by referring to the numbered references within the figure.

1. The process begins using a CAD/CAM system to design the product and provide the product design information to the manufacturing system. Once the product design phase is complete, the various system components (robots and automated machinery) are programmed to perform their respective manufacturing tasks.
2. The production process begins when a computer controlled parts carrier delivers raw materials to the production line. The parts carrier has been loaded automatically by a robot in the parts storage area. The carrier follows a path to the production line that is defined by a wire buried in the factory floor.
3. When the raw materials arrive, a pick-and-place robot unloads them from the carrier and places them on a conveyor.
4. An assembly robot assembles the parts as they move down the conveyor.
5. A welding robot performs a welding operation to join the parts together.
6. The finished product is then inspected by a computerized inspection system. In addition to vision, the inspection system might be designed to perform mechanical and electrical tests on the product. Mechanical shock and pull tests might be used to inspect for the strength of the weld seams. Electrical resistivity tests can also be performed to inspect for weld integrity.
7. Finally, a pick-and-place robot places any parts that have been rejected into a bin for engineering disposition. Acceptable parts are placed onto a carrier for transportation to a product storage area where they will be unloaded and stored by another robot.
8. Remote terminals form a data acquisition system, or DAS, that allow engineering and management to monitor the production process. Such a system provides up-to-the-minute production control and quality control information.

The largest applications for flexible manufacturing systems is for small volume, or **batch**, processes. Over seventy-five percent of all machined parts produced are in batches of 100 or less. Without FMS, a manufacturing process requires that machines be dedicated to a specific task, as defined by the product. When the product changes, the machine fixtures and tooling must be rebuilt or replaced according to the new product definition. However, with FMS, product changes are incorporated by simply changing the system software.

An FMS can produce a small batch of a product as economically and more efficiently than a conventional production line developed to produce several thousands of parts. In addition, the manufacturer with an FMS is much more diversified. Such a manufacturer can maintain a competitive edge by changing the product almost instantly as market needs change, without having to invest in costly new tooling. Moreover, the manufacturer can move rapidly into new product lines and drop old ones just as rapidly if the market so dictates.

Although an FMS is not cheap, its flexibility, productivity, and labor saving features provide a relatively short term return on the original investment. This is especially true for small production run, or batch, processes. Several FMS processes are already in place, mostly in Japan. With increased emphasis on productivity and a need to maintain a competitive edge in the world market, the future also looks extremely bright for FMS in U.S. industry.

## Commercial Aviation Applications

The aviation industry has made extensive use of microprocessors, especially in new aircraft designs and avionics (electronic aviation instrumentation). Much of the microprocessor applications technology in commercial aviation has evolved from military and space programs. However, the real impetus for using microprocessors in new aircraft resulted from a need to make them safer and more fuel efficient.

Before the microprocessor, it was not feasible to install digital computers aboard an airliner due to their size and cost. Now, with microprocessors, computer intelligence can be economically dedicated to perform several individual tasks aboard an aircraft. As a result, the aircraft is safer, more fuel efficient, and the pilot is free from the many manual chores required before use of the microprocessor.

The aircraft illustration in Figure 10-18 shows several ways in which microprocessors are being used in newer generation aircraft. Notice that individual microprocessors are dedicated to specific tasks such as navigation, passenger comfort, engine control, and aerodynamic surface control. The dedicated microprocessors are linked to a central microprocessor for overall system monitoring, supervision, and control. In many cases, the dedicated devices are 8-bit microprocessors, and the system device is a 16-bit microprocessor. In addition, the systems are often redundant for increased reliability and safety.

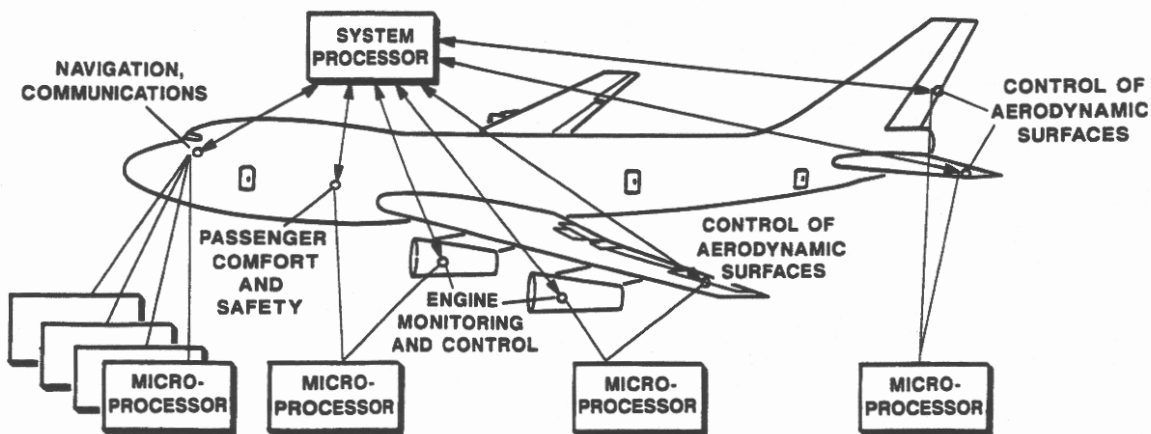


Figure 10-18

Dedicated applications of a microprocessor aboard a commercial airliner.

Now, let's take a look at how microprocessor technology is being applied to a specific aircraft, the Boeing 757/767. Figure 10-19 shows a block diagram of the 757/767 microprocessor-based flight management system. The system consists of a central flight management computer, or FMC, and several subsystem computers.

At the heart of the FMC is a 16-bit microprocessor, the Texas Instruments TMS9900. The subsystems use high performance 8-bit microprocessors such as the Intel 8085. In addition, numerous 8-bit microprocessors are dedicated to sensing and controlling various part of the aircraft. These dedicated devices link to the central flight management computer. Moreover, there are two FMCs—one for the pilot and one for the copilot. The aircraft can be flown with either of the FMCs, or it can be flown manually.

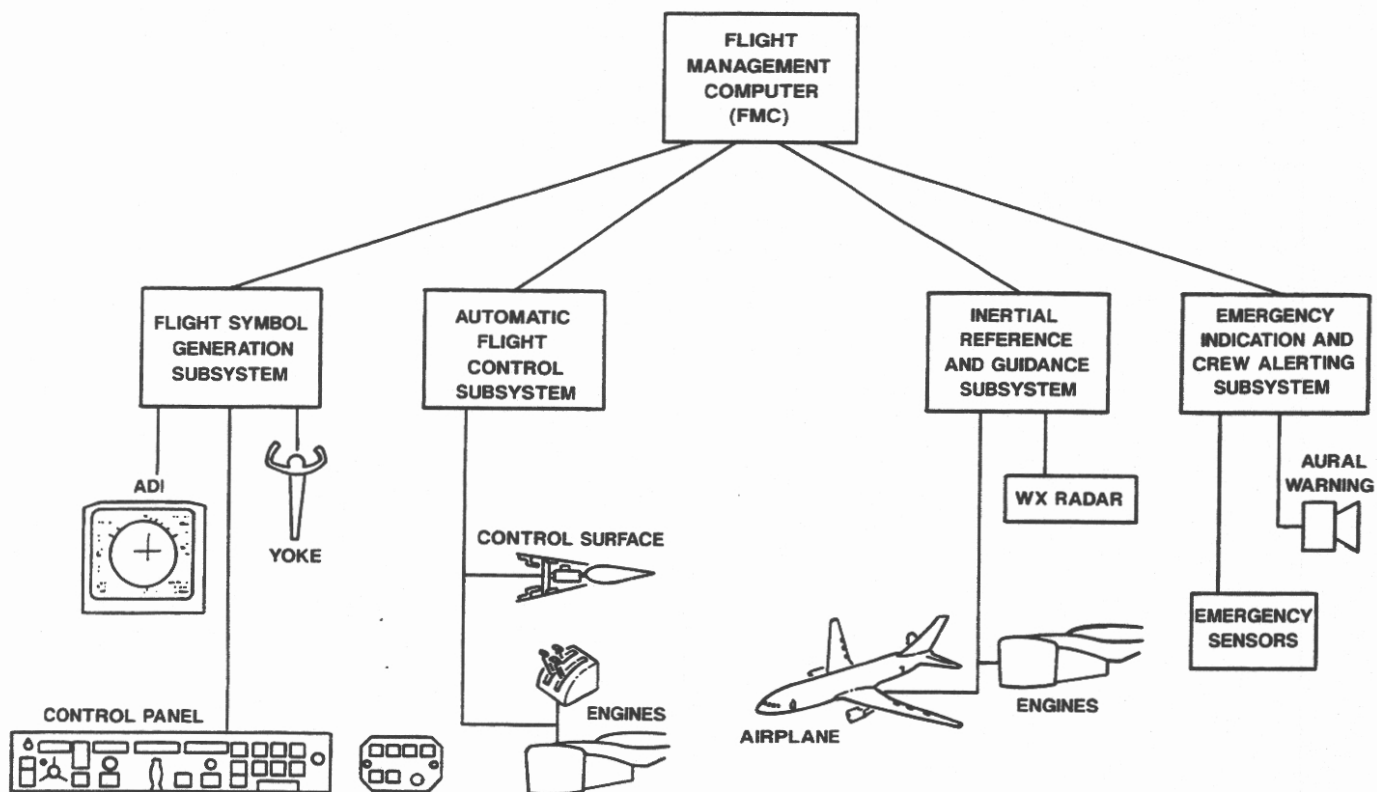


Figure 10-19  
A block diagram of the Boeing 757/767  
microprocessor-based flight management system.

The 757/767 instrument panel is illustrated in Figure 10-20. Notice that there are no less than six high-resolution CRT color-graphic displays. These displays provide flight status and control information at the push of a button. For example, the horizontal situation indicator, or HSI, CRT displays a map of the aircraft's flight path along with an overlay of the on-board weather radar display. Such a display allows the flight crew to determine the safest and most economical flight path.

Another CRT is used as an electronic attitude direction indicator, or ADI, which is a standard instrument used to indicate the aircraft's pitch and roll. Before the 757/767, the ADI was an electromechanical type instrument. Now, it is a CRT display in the cockpit. Other CRTs display navigation information, engine status, landing gear status, and emergency information such as smoke or fires in the cargo bay.

The 757/767 flight management system is so advanced that it can literally fly the aircraft from takeoff to touchdown. In fact, as a passenger, you won't be able to tell if the pilot or the microprocessors are flying the airplane. The system allows the pilot to become a *flight manager*. The "real" pilot is the microprocessor-based flight management system.

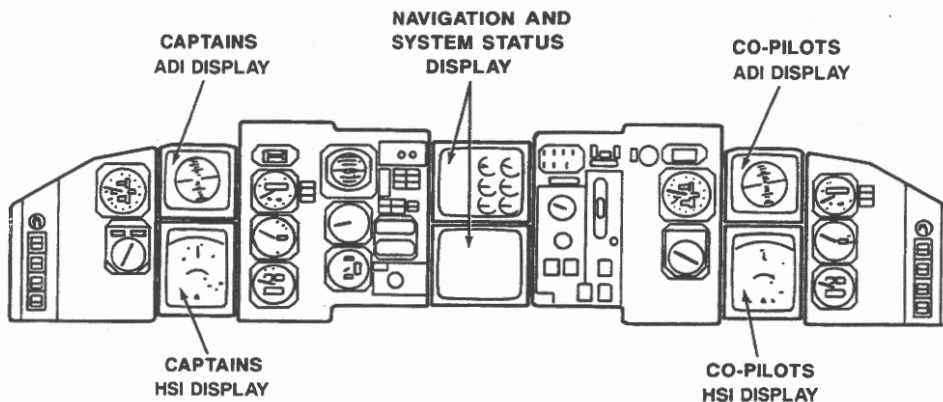


Figure 10-20  
The Boeing 757/767 cockpit instrument panel.

## Medical Applications

The medical industry is making extensive use of microprocessor technology. Microprocessors have literally generated a whole new breath of life into the field of biomedical engineering. Typical applications range from simple smart thermometers to implanting microprocessor circuits that restore movement to paralyzed limbs. Microprocessor technology is also helping the blind to see through image processing. The possible applications of microprocessor technology in the medical field are almost endless. Indeed, we all stand to benefit from this technology.

A typical medical application of microprocessors is the portable bedside monitoring station illustrated in Figure 10-21. Using computers to collect and analyze patient data is not new. However, before the microprocessor, patient data had to be transmitted over a considerable distance to a central computer for analysis. This resulted in poor response time since, in most cases, the central computer was also required to perform other tasks. Response time is critical, especially in life-and-death situations. Now, with microprocessor technology, a microcomputer can be dedicated solely to the patient monitoring task. Besides providing better response times, the microcomputer has significantly reduced the cost of such systems.

The portable bedside microcomputer unit illustrated in Figure 10-21 can be used to monitor up to three patients. The microprocessor monitors respiration by collecting air flow, air pressure, oxygen, and carbon dioxide data. Cardiovascular information is collected by monitoring temperature, pulse rate, and blood pressure. The data is input to the microprocessor via sensors and transducers that are attached directly or indirectly to the patient. Many of the sensor/transducer principles described in Units 7 and 8 of this course are applied.

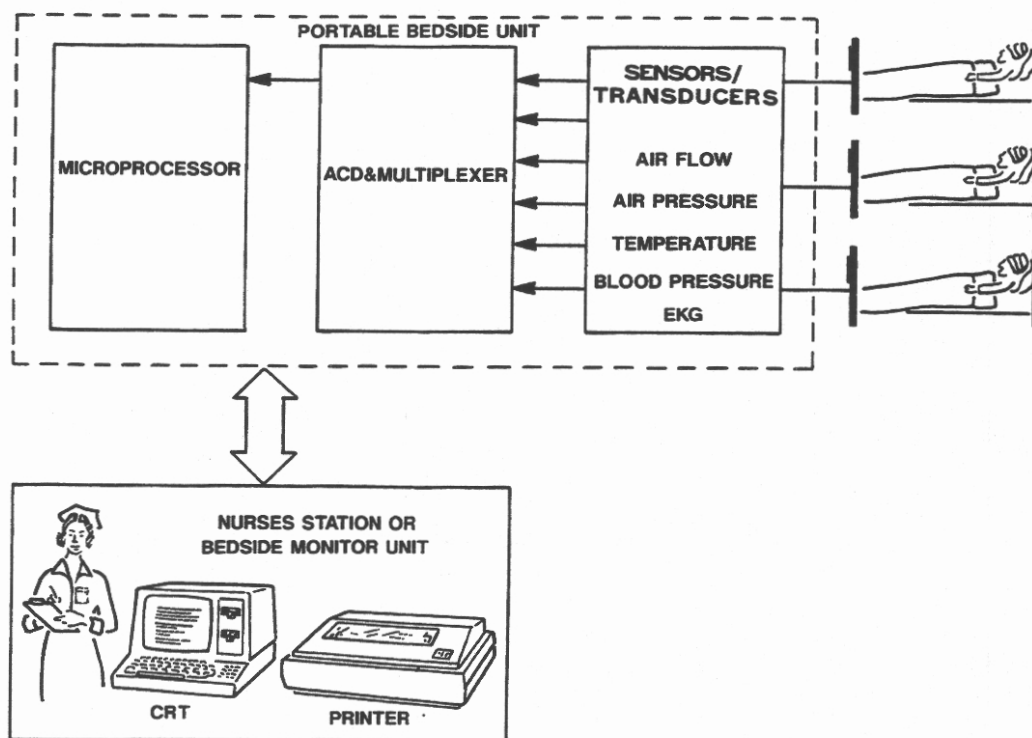


Figure 10-21

A microprocessor-based patient monitoring system.

Analog data from the sensors and transducers is multiplexed and converted to digital data by an analog-to-digital converter circuit. The microprocessor is programmed to analyze the patient data and generate a video display of the patient status in medical terms. In addition, a printer generates a hard copy of the patient status, thereby producing a medical history of the patient. The video display and printer can be located at the patient's bedside, or in a central nurses' station.

Another feature of the system is that the microprocessor can be programmed to set off an alarm if it detects an emergency condition. Furthermore, a doctor can use this microcomputer keyboard and display as a general purpose computer to perform additional medical analysis in life-threatening situations.

A future improvement of the above system is already being developed. Several companies are developing a smart hospital bed that contains all the required monitoring circuitry, including a microprocessor circuit that is preprogrammed to perform the monitoring and analysis tasks. A video terminal and/or printer is connected to the smart bed to allow doctors and nurses to monitor the patient status.



## Self-Test Review

9. List the three major sections of an intelligent robot. \_\_\_\_\_  
\_\_\_\_\_
10. Define sensory feedback. \_\_\_\_\_  
\_\_\_\_\_
11. Define CAD/CAM. \_\_\_\_\_  
\_\_\_\_\_
12. List at least three advantages of using a CAD/CAM workstation. \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_
13. Name the major components of a typical CAD/CAM workstation. \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_
14. Define FMS. \_\_\_\_\_  
\_\_\_\_\_
15. List at least five applications for a microprocessor aboard an aircraft. \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_
16. Describe the operational features of a microprocessor-based patient monitoring system. \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

## Answers

9. The three major sections of an intelligent robot are the power supply, controller, and manipulator.
10. Sensory feedback in a robot allows the robot to detect external conditions and make assembly decisions.
11. CAD/CAM is an acronym for Computer-Aided Design/Computer-Aided Manufacturing. It is the technology of using computers in the production process, from early product design to final assembly and testing.
12. Advantages of using a CAD/CAM system in a production process include:
  - Increased productivity.
  - Decreased manufacturing cost.
  - Increased product quality.
  - Earlier product introduction into the marketplace.
  - Design flexibility.
13. A typical CAD/CAM work station includes a graphics CRT terminal, keyboard, graphic tablet, light pen, and a printer/plotter.
14. FMS is an acronym for a Flexible Manufacturing System. Such a system incorporates CAD/CAM and robotics to produce a completely automated manufacturing process.
15. Microprocessor applications aboard an aircraft include:
  - Navigation.
  - Passenger/crew comfort.
  - Aerodynamic surface control.
  - Engine monitoring and control.
  - Communications.
  - Safety and emergency alert.
  - Landing gear monitoring and control.
16. A microprocessor-based patient monitoring system is typically a bedside unit that monitors patient respiration, temperature, pulse rate, blood pressure, etc. The system is programmed to analyze the patient data and generate a video display and hard copy printout of the patient status in medical terms. In addition, the system can be programmed to set off an alarm if it detects an emergency condition. Medical personnel can also use the system as a general purpose computer to perform medical analysis.

## BUSINESS APPLICATIONS

Clearly, business is one of the largest benefactors of microprocessor technology. What began with small electronic calculators has evolved into complete microprocessor-based business computer systems. Small business has benefited the most, since tremendous computing power is now affordable. Inexpensive microcomputer business systems allow accounting, billing, and payroll. Before the microprocessor, these computerized luxuries were only available to large businesses that could afford expensive mainframe computer systems. As a result of microprocessor technology, small businesses are better prepared to compete with large businesses in the marketplace.

You might think that applications of microprocessors in business are limited to systems applications, such as business computing systems and word processors. On the contrary, many business applications are dedicated applications. These applications include smart typewriters, cash registers, and copiers, just to mention a few. Now, let's take a closer look at several of the more common business applications of microprocessor technology.

### Small Business Systems and Word Processors

A typical small business microcomputer system is pictured in Figure 10-22. As you can see, such a system usually consists of a keyboard/video display unit, mass storage disk system, and a hard copy printer.

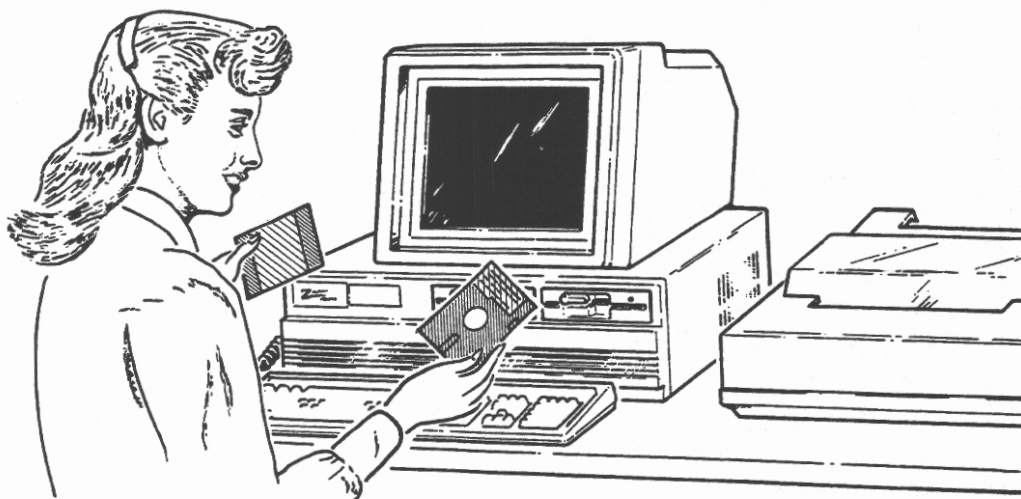


Figure 10-22  
A typical small business microcomputing system.

The keyboard is a standard typewriter keyboard that may include a separate numeric key pad. The video display usually provides a display of 24 lines with 80 characters per line. High resolution graphics may or may not be provided, depending on the system.

The mass storage unit is usually a 3 1/2" or 5 1/4" floppy disk system capable of storing several hundred thousand bytes of information. One or more disk drive units may be required. Hard disk systems, such as a Winchester disk, store more information than floppy disks but are more expensive. However, hard disks are beginning to replace floppy disks in the business market.

Finally, the system printer must be capable of producing a hard copy of "letter quality." There are numerous types of printers available. The more popular business printers are daisy wheel and laser printers. Both type printers are economical and produce excellent printing quality.

The business system usually contains a resident software package, like BASIC or Pascal, for general purpose programming. In addition, most manufacturers support their systems with a complete line of disk software. Available software might include COBOL, FORTRAN, word processing packages, and various business related software for payroll, accounting, billing and marketing.

Of particular interest to business are the word processing features of a microcomputer business system. Word processing makes creation of documents faster and easier. Such systems allow an experienced secretary to enter, edit, and print a letter quality document in much less time than with a conventional typewriter. Furthermore, important documents can be permanently saved on disk, thereby eliminating the need for document filing space and equipment.

## Copiers

Microprocessor technology has revolutionized the copier industry. Microprocessors have made copiers smaller, more efficient, and more economical, while producing better quality copies. Moreover, microprocessor-based copiers, like the one pictured in Figure 10-23, provide features such as document enlargement and reduction, automatic document feed, and sorting.

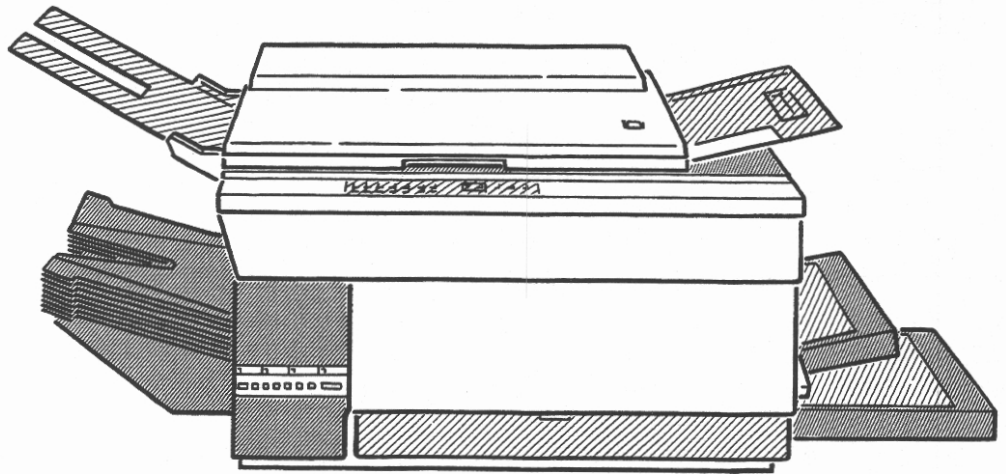


Figure 10-23  
An intelligent copier.

A block diagram of a microprocessor-based copier circuit is shown in Figure 10-24. Temperature sensors, light sensors, and position sensors generate signals during the copying operation which are multiplexed into the microprocessor. The microprocessor makes control decisions based on the sensed conditions and issues commands to control devices such as motors, solenoids, relays, heaters, and so on. Drivers are used to amplify the low-level microprocessor outputs to control the higher power loads. All of the microprocessor control programs are located in ROM, EPROM or EEPROM. Optional features can then be added by simply substituting different ROMs, EPROMs or EEPROMs.

Does the above system look and sound familiar? It should, since it applies many of the closed-loop sense and control principles discussed in this course.

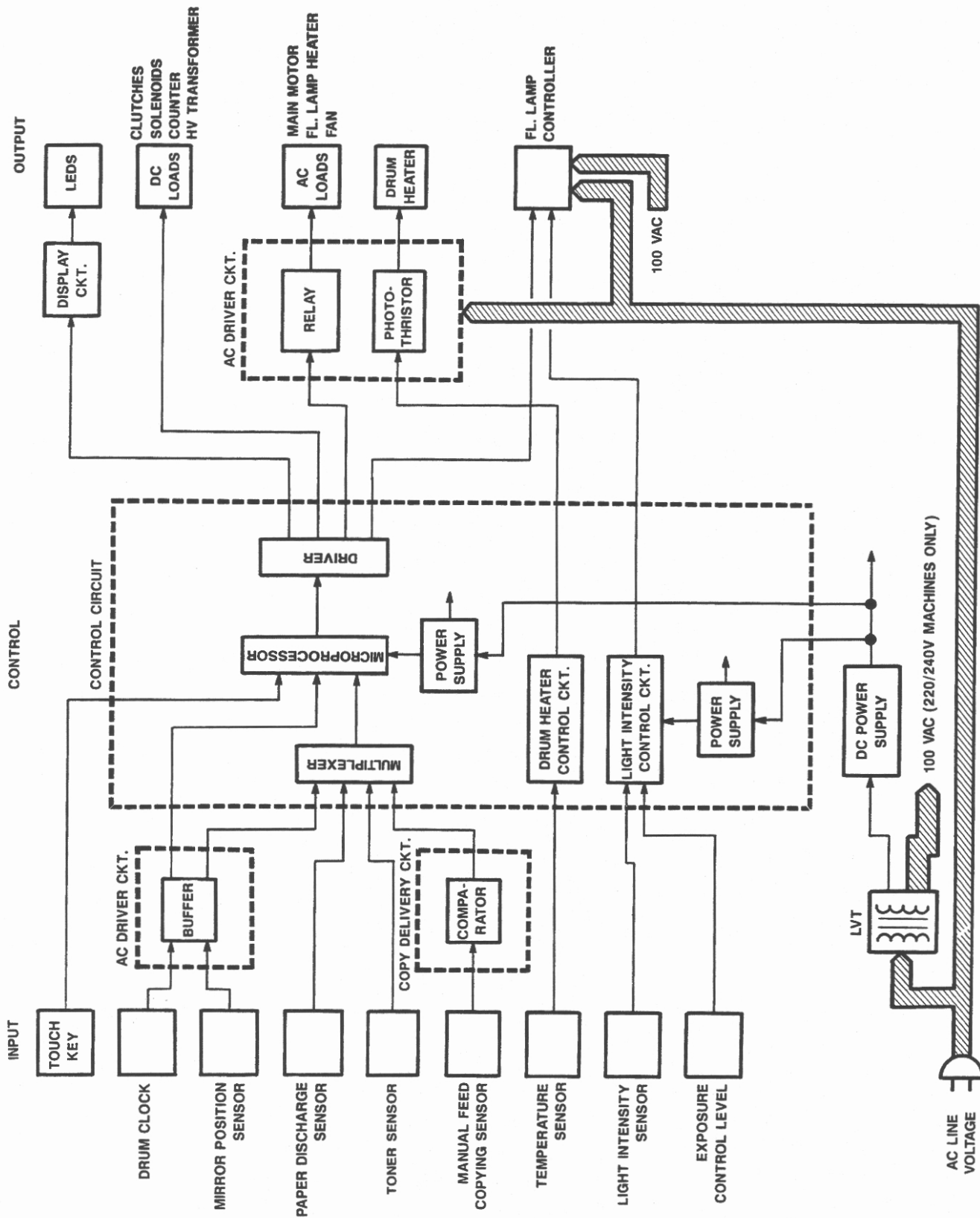


Figure 10-24  
A block diagram of a typical intelligent photo-copier

## Cash Registers and Inventory Control

Just like word processors and copiers have revolutionized the office industry, electronic cash registers have revolutionized the retail industry. A typical microprocessor-based electronic cash register is pictured in Figure 10-25. Such cash registers are smaller, lighter, quieter, more efficient, and less expensive than their old electromechanical counterparts. Furthermore, electronic cash registers provide increased accounting abilities. Typical features of an electronic cash register may include:

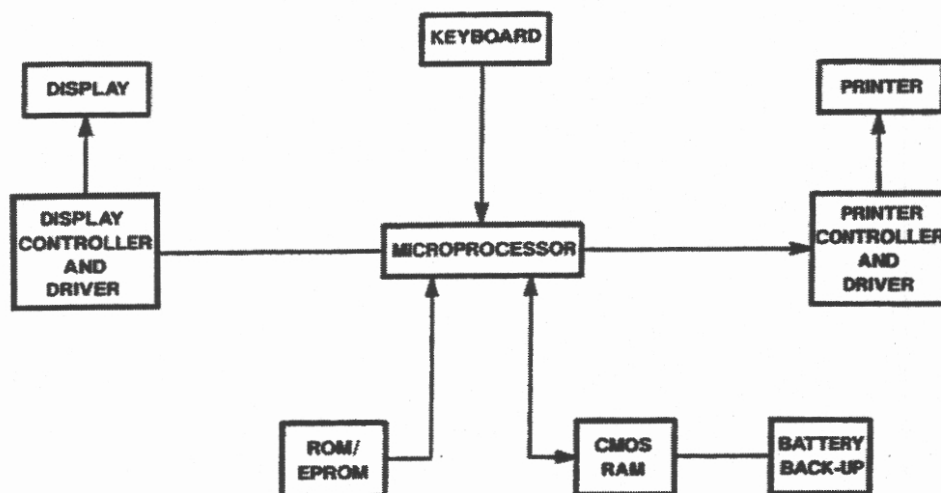
- Department totals by dollar amount, and percentage of total sales.
- A 24-hour time-of-day clock to monitor business activity according to specific time periods.
- Sales clerk identification and security.
- Automatic discount calculation for sales items.
- Automatic sales tax calculation and totals.
- Multiplication key for multiple quantities of a given item.
- Data retention during power loss.
- Programmability for sales discounts and taxes.
- Inventory control system interface.



Figure 10-25  
An electronic cash register, or ECR.



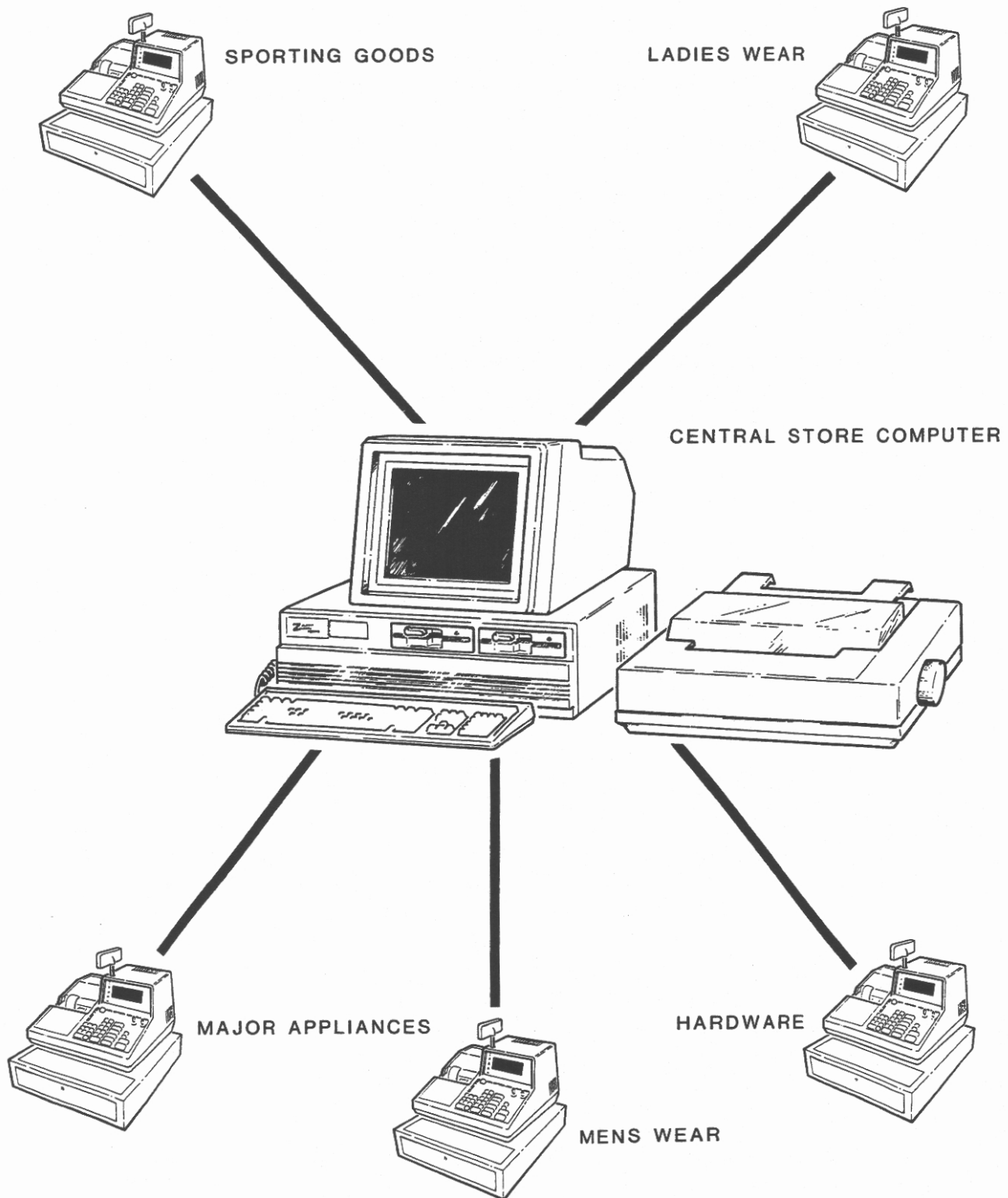
Most of the above features are possible, since the electronic cash register is microprocessor-based. A block diagram of a typical electronic cash register circuit is shown in Figure 10-26. A microprocessor is used to read the keyboard and control the activities of the printer, memory totals in RAM, cash drawer, display, and real time clock. The system programs are permanently stored in ROM or EPROM. EPROMs allow the manufacturer to tailor the features of a cash register to a given retailer. In many cases, the system contains CMOS RAM with a battery backup unit. This feature allows sales and tax data to be retained in the case of a power failure.



**Figure 10-26**

A block diagram of a typical cash register circuit.

Many times, the electronic cash register is linked to a central store computer as illustrated in Figure 10-27. In this case, it is called a point-of-sale terminal. The point-of-sale terminal handles all the local sales transactions and prints the customer receipt. In addition, sales data is transmitted to the central store computer for analysis and inventory control. This allows the store manager to monitor the sales activity in several departments. Moreover, the central computer can be programmed to generate periodic inventory reports and warehouse orders.



**Figure 10-27**  
Electronic cash registers act as point of sales terminals  
which link into a central store computer.

In large retail chains, the individual store computers are linked to a regional computer. Just like an individual store manager can monitor departmental sales activity and inventory, a regional sales manager can monitor the sales and inventory activity of individual stores. Furthermore, the regional computer can be programmed to provide credit verification and handle credit card transactions at the point-of-sale terminal.

In national retail chains, the regional computers are linked to a national computer for sales and inventory management on a national basis. As you can see, the simple microprocessor-based electronic cash register can be expanded into a large multiprocessor system as the need dictates.

In many stores, especially grocery stores, the point-of-sale terminal includes an optical scanning system as illustrated in Figure 10-28. The scanner/register reads a universal product bar code, or UPC, that is printed on the product label. The product code is received by a microprocessor controller which uses a memory look-up table to send back pricing information to the terminal. The price of the product, as well as any sale discounts, are programmed into the system. All the sales clerk needs to do is to scan the product bar code. The system then returns the price and product description.

A scanning system is especially attractive for large volume check-out terminals, as in grocery stores. Aside from product check-out, the system has a central store computer that is programmed for inventory control. The store or department manager no longer needs to make out a lengthy warehouse order, since the store computer is programmed to generate warehouse orders according to given inventory limits. In fact, the store computer can be linked directly to the warehouse computer to produce a completely automated product ordering system. The store manager only needs to periodically update the inventory data base as product needs change.

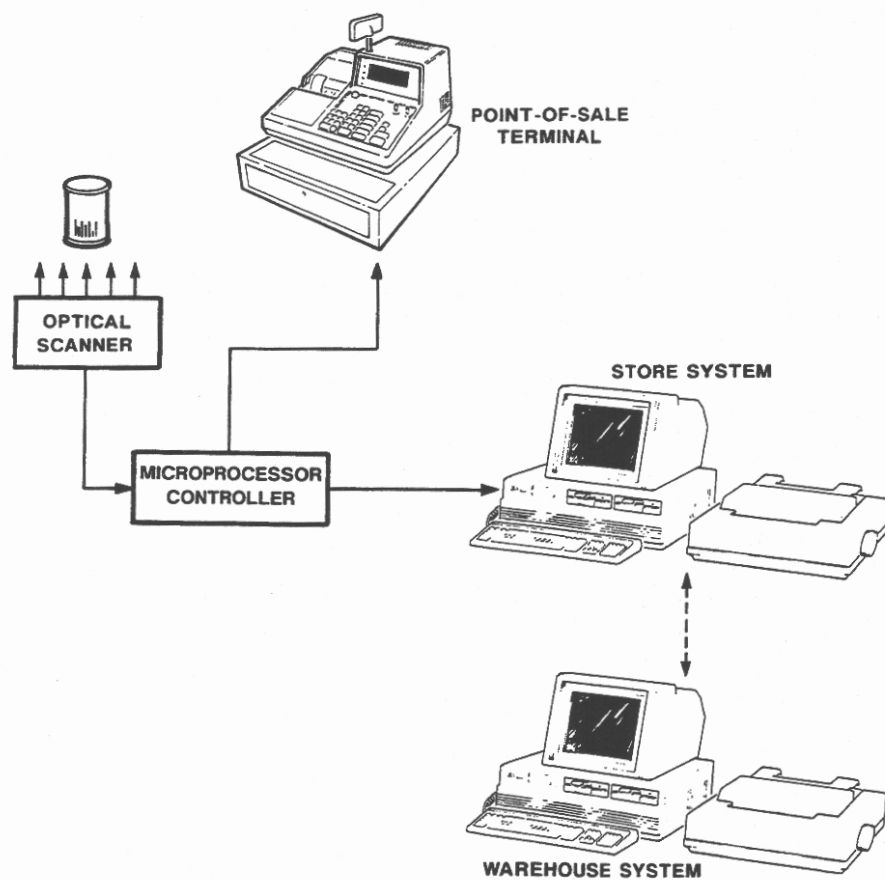


Figure 10-28

A typical microprocessor-based product scanning and inventory control system.

## Self-Test Review

17. List the three hardware items that are required to form a small business microcomputer system. \_\_\_\_\_  
\_\_\_\_\_
18. What type of printer is very popular with small business systems because of its cost and printing quality. \_\_\_\_\_  
\_\_\_\_\_
19. What type of sensors are generally found in a microprocessor-based copier? \_\_\_\_\_  
\_\_\_\_\_
20. State the advantages of an electronic cash register over an electro-mechanical cash register. \_\_\_\_\_  
\_\_\_\_\_
21. Explain the difference between an electronic cash register and a point-of-sale terminal. \_\_\_\_\_  
\_\_\_\_\_
22. Describe a microprocessor-based product scanning and inventory control system. \_\_\_\_\_  
\_\_\_\_\_

## Answers

17. The three hardware items required to form a small business microcomputer system are a keyboard/video display unit, a mass storage disk system, and a hard copy printer.
18. Daisy wheel and laser printers are very popular for small business systems because they are economical and produce excellent printing quality.
19. Temperature, light, and position sensors are generally found in a microprocessor-based copier.
20. An electronic cash register is smaller, lighter, quieter, more efficient, less expensive, and contains more accounting features than an electromechanical cash register.
21. A point-of-sale terminal is an electronic cash register that is linked to a central computer.
22. A microprocessor-based product scanning system consists of an optical product scanner, microprocessor controller, and a point-of-sale terminal. The microprocessor controller reads the product code via the scanner and sends back product and pricing information to the terminal. The microprocessor controller is also linked to a central store computer which, in turn, is linked to a warehouse computer for automated product ordering and inventory control.

## UNIT SUMMARY

The microprocessor applications industry has taken two directions: dedicated applications for control, and systems applications for data processing. Complete self-contained single-chip microcomputers have been developed for dedicated applications. Sophisticated microprocessor CPUs, like the 68000 and 80386, have been developed to support systems applications.

Most microprocessor applications fall into one of three major categories: consumer, industrial, and business. Consumer applications of microprocessors included automobiles, PCs, and appliances, just to mention a few. Industrial applications of microprocessors include CAD/CAM, robotics, data acquisition, process control, aviation, and medicine. The ultimate automated manufacturing system is the flexible manufacturing system, or FMS, which employs microprocessor technology from initial product design to final assembly and inspection. Business is one of the largest benefactors of microprocessor technology. What began with electronic calculators has evolved into complete microprocessor-based business computer systems and networks. Such systems are used for market forecasting, planning, research, accounting, billing, payroll, and inventory control. These are just a few of the more common business applications of microprocessors.

Congratulations! You have just completed this comprehensive course on microprocessor interfacing and applications. We hope you have enjoyed your learning journey through this fascinating technology. Now it's up to you to apply what you have learned. Remember, the possible applications of microprocessor technology are only limited by your imagination.

# INDEX



## INDEX

## A

## A/D CONVERSION

- ADC Devices 6-40, 47
- Applications 6-57, 70
- Interfacing 6-40, 53
- V/F Converters 6-47, 53

## Address Decoding 1-8

## Analog Multiplexing 6-34

- In a Data Acquisition System 6-59, 60

## ANALOG CONVERSION

- A/D 6-40, 73
- D/A 6-6, 36

## ANALOG SENSING

- In Industrial Control Systems 6-66, 70

## Appliance Applications 10-11, 12

## ASCII 4-10

- Code Table 4-12

## Automobile Applications 10-6, 10

## Aviation Applications 10-30, 33

- Flight Management Computer 10-32, 33

## B

## Buffering 1-12, 15

## BUSES

- Types 1-6

## C

## CAD/CAM 10-27, 28

## Capacitive Sensors 8-10, 17

- Position 8-12, 16

- Proximity 8-17

## Cash Registers 10-42, 47

## Comparator 6-26, 27

## Compression Strain 8-36, 37

## CONVERSION

- A/D 6-40, 73
- D/A 6-6, 36
- Hardware Parallel/Serial 4-30
- Software Parallel/Serial 4-17

## Copiers 10-40, 42

## D

## D/A CONVERSION

- Applications 6-13, 36
- Interfacing 6-6, 10

## Data Acquisition 6-58, 70

## DC MOTORS

- See Motors

## Debouncing Switches 1-35, 42, 43

## DECODING

- Address 1-8
- Full 1-9
- ICs 1-11
- Keyboards 2-24, 27
- Partial 1-10
- Switches 1-35

## Difference Amplifier 6-28

## DISPLAYS

- Interfacing 1-46, 55
- Multiplexing 1-54
- 7-Segment 1-46

## Dynamic RAM 4-13, 2-5

## E

## Eddy Current Probe 8-25, 26

## EPROM

- Architecture 2-30, 31
- Programming 2-34

## F

## Flexible Manufacturing Systems 10-29, 30

## Force Sensors 8-36, 53

- Load Cells 8-50, 53

- Piezoelectric Strain Gages 8-50

- Resistive Strain Gages 8-38, 48

- Semiconductor Strain Gages 8-48, 50

## Frequency Modulation (FSK) 4-9

## Full Duplex 4-13, 14

## G

## Gage Factor 8-41

**H**

Half Duplex 4-13, 14

**HANDSHAKE**

With A/D Converters 6-42, 47

**I**

Inductive Sensors 8-18, 26

LVDT 8-21, 23

Position 8-19, 26

Proximity 8-25, 26

RVDT 8-24

Industrial Control Systems 6-66, 70

Instrumentation 6-57

Instrumentation Amplifier 8-46

Integrated Optical Sensors 7-58, 63

Application Circuits 7-60, 63

Characteristics 7-59

Construction and operation 7-58

Interfacing 1-5

Address decoding 1-7

Busses 1-5

Displays 1-52

EPROM 2-29

RAM 2-5

Tri-state buffering 1-11

8085 Address lines 1-20

8085 Control lines 1-22

8085 Data lines 1-19

Interrupts 1-23

Intr (Interrupt request) 1-24

RESET 1-24

TRAP/RST Interrupts 1-28

Inventory Control 10-42, 47

I/O Execution

Control 3-30

Direct I/O 1-32

Memory-Mapped I/O 1-33

IREDA 7-58; 9-14, 15

**K**

Keyboards 1-41, 42, 43, 44, 45, 46

**L****LATCH**

Addressable 1-50, 54

**LDR**

See Photoconductive Optical Sensors  
Load Cells 8-50, 53

Application Circuits 8-53

Characteristics 8-52

Construction and Operation 8-51, 52

LVDT 8-21, 23

**M**

Magnetic Sensors 8-26, 32

Hall-Effect 8-29, 30

Position and Proximity 8-27, 32

Reed Switches 8-27, 28

Mark Space 4-7, 8

Medical Applications 10-33, 35

Memory 1-6, 7; 2-1, 54

EPROM 2-29

RAM 2-5

Microwave Oven 10-6, 10

**MOTORS**

Control and Positioning Using a D/A Converter  
6-24, 30

Control Using Pulse-Width Modulation 9-37, 42

Control of Wound-Field Motors 9-42, 44

Stepper 9-45, 58

MUART 4-36 to 4-65

**O**

Open Collector Drivers 9-7, 8

Optical Encoders 7-62, 63

Optical Interrupter Module 7-58, 63

Optical Reflector Module 7-58, 63

Optical Sensors 7-41, 69

Integrated Sensors 7-58, 63

Photoconductive Sensors 7-41

Photovoltaic Sensors 7-47, 54

Summary 7-64

Optocouplers 9-14, 16

Optoisolators 9-14, 16

**P**

- Period Measurement 5-50, 52
- Peripheral Power Drivers 9-13
- Personal Computers 10-18, 20
- Photoconductive Optical Sensors 7-41, 47
  - Application Circuits 7-45, 47
  - Characteristics 7-43, 44
  - Construction and Operation 7-41, 42
- Photodiodes 7-47, 54
  - Application Circuits 7-52, 54
  - Array 7-53
  - Characteristics 7-52
  - Construction and Operation 7-48, 49
  - Photoconductive Mode 7-51
  - Photovoltaic Mode 7-49, 50
- Phototransistors 7-54, 58
  - Application Circuits 7-56, 58
  - Characteristics 7-56
  - Construction and Operation 7-54, 55
- Photovoltaic Optical Sensors 7-47, 54
  - Photodiodes 7-47, 54
  - Phototransistors 7-54, 58
- Piezoelectric Strain Gages 8-50
- Polled Control of I/O Operations 3-46
- Position encoder 6-29, 30
- Position and Proximity Sensors 8-6, 32
  - Capacitive 8-10, 17
  - Inductive 8-18, 26
  - Magnetic 8-26, 31
  - Potentiometric 8-6, 9
  - Summary 8-32
- Potentiometric Position Sensors 8-6, 9
- Power Transistors 9-9, 10, 12
  - Bipolar 9-9, 10
  - MOSFET 9-12
- Preset Indexer 9-58
- PROCESS CONTROL
  - Using a D/A Converter 6-30, 36
- Programmable Gain Amplifier and Attenuator 6-22, 24
- Programmable Parallel I/O Ports 3-5, 7, 8, 13, 15
- PROGRAMMABLE TIMERS
  - See Timers, Programmable

**PTM**

- Addressing 5-23, 25
- Continuous Waveform Generation 5-34, 42
- Initialization 5-26, 27
- Interfacing 5-23, 25
- Interrupt Generation 5-31, 34
- I/O Diagram 5-10
- One-Shot Pulse Generation 5-43, 50
- Period Measurement 5-50, 52
- Pulse Measurement 5-53, 55
- Registers 5-11, 19
- Pulse Measurement 5-53, 55
- Pulse Width Modulation 9-37, 42

**R****RAM**

- Dynamic 2-12
- Interfacing 2-6
- Refresh 2-18
- Static 2-6
- Reference Junction 7-25, 26
- Relays 9-25, 31
  - Electro-Mechanical 9-28, 29
  - Reed 9-29, 31
  - Solid State 9-26, 27
- Resistive Strain Gages 8-38, 48
  - Application Circuits 8-43, 48
  - Characteristics 8-43
  - Construction and Operation 8-39, 42
- Robotics 10-23, 26
- RTD Temperature Sensors 7-7, 14
  - Application Circuits 7-12, 14
  - Characteristics 7-10, 11
  - Construction and Operation 7-8, 10

**S**

- Sample/Hold 6-34, 35, 61
  - In a Data Acquisition System 6-61, 65
- SCRs 9-17, 19
- Seebeck Voltage 7-19
- Semiconductor Junction Temperature Sensors 7-27, 30

Semiconductor Strain Gages 8-48, 50

Application Circuits 8-49, 50

Characteristics 8-49

Construction and Operation 8-48

Serial Data Communications 4-7

ASCII 4-10, 4-11

Asynchronous 4-10

Channeling 4-13

Servo Amplifier 6-24

Shear Strain 8-36, 37

Simplex 4-13

Snubber Circuit 9-27

Solar Cell 7-47, 54

Solenoids 9-32

Static RAM 2-6, 12

Stepper Motors 9-45, 58

Bipolar 9-46, 48

Control of 9-50, 58

Unipolar 9-48, 50

Strain 8-36, 37, 45

STRAIN GAGES

Piezoelectric 8-50

Resistive 8-38, 48

Semiconductor 8-48, 50

Stress 8-36

SWITCH

Debouncing 1-35, 42, 43

Decoding 1-35

Detecting Closure 1-34, 35, 42

Keyboards 1-41 to 1-46

## T

Temperature Coefficient of Linear Expansion 7-31

Temperature Sensors 7-6, 36

Semiconductor Sensors 7-27, 30

Summary 7-36

Thermoelectric Sensors 7-18, 26

Thermoresistive Sensors 7-6, 18

Thermoswitches 7-31, 36

Tensile Strain 8-36, 37

Thermistor Temperature Sensors 7-14, 18

Application Circuits 7-17, 18

Characteristics 7-16, 17

Construction and Operation 7-14, 15

Thermocouple Temperature Sensors 7-18, 26

Application Circuits 7-22, 26

Characteristics 7-20, 21

Construction and Operation 7-19

Thermoswitches 7-31, 36

Thyristors 9-16, 25

Control Circuits 9-20, 25

SCRs 9-17, 19

TRIACs 9-19, 20

Transistor Arrays 9-10, 11

Timers, Programmable 5-1, 85

General Concepts 5-7, 9

Initialization 5-8

6840 PTM 5-10, 55

TIMING

Instruction 1-27, 28

Program 1-28, 30

TRIACs 9-19, 20

## V

V/F Converters 6-47, 53

Interfacing via Hardware 6-48, 50

Interfacing via Software 6-51, 52

Remote Sensing 6-53

## W

WAVEFORM GENERATION

Via a D/A Converter 6-14, 19

Via a PTM 5-34, 50

Weather Computer 10-13, 15

Word Processors 10-38, 39

Wound Field DC Motors 9-42, 44

## X

X-Y Displays 6-20, 21

